REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		ANNUAL 01 Aug 92 TO 31 Jul 93

4. TITLE AND SUBTITLE

APPLICATION OF DIFFERENTIAL INVERSION TO DMSP MICROWAVE SOUNDER DATA

5. FUNDING NUMBERS

F49620-92-J-0444
61102F
2310
CS

6. AUTHOR(S)

Dr Robert G. Hohlfeld

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Dept of Engineering
Boston University
44 Cummington Street
Boston, MA 02215

8. PERFORMING ORGANIZATION REPORT NUMBER

AFOSR-TR- 94 0201

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFOSR/NL
110 Duncan Avenue, Suite B115
Bolling AFB DC 20332-0001

Maj Kroll

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

94-12045

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release;
distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Differential Inversion (DI) is a novel approach to the solution of the atmospheric temperature sounding problem which was developed by Dr J.I.F. King of the Air Force's Phillips Laboratory. Before the present research, DI has been applied only to infrared radiance data sets, such as from TOVS/HIRS. In this report I describe the progress made in the first year of a research program to apply DI to microwave radiance data from the SSM/T sounder on the DMSP satellites. The ultimate objectives of this research are to establish effective DI sounding algorithms for the microwave spectral region and to extend DI in directions which increase its utility as a practical sounding algorithm. DI has many attractive features in this application, including itsclose coupling to the physical formulation of the temperature sounding problem, its freedom from the necessity of using an a priori temperature profile, and its high level of computational efficiency. At present, we have understood the practical difficulties characteristic of temperature sounding in the microwave spectral region (as these apply to DI), and have produced software yielding temperature profiles of a convincing meteorological character. Research currently planned for the second year will continue with a detailed comparision with ground truth data. Theoretical advances in the past

14. SUBJECT TERMS

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
(U)	(U)	(U)	(UL)

Application of Differential Inversion to DMSP Microwave Sounder Data: First Year Progress Report on AFOSR Grant F49620-92-J-0444

Prof. Robert G. Hohlfeld*

Abstract

Differential Inversion (DI) is a novel approach to the solution of the atmospheric temperature sounding problem which was developed by Dr. J. I. F. King of the Air Force's Phillips Laboratory. Before the present research, DI has been applied only to infrared radiance data sets, such as from TOVS/HIRS. In this report I describe the progress made in the first year of a research program to apply DI to microwave radiance data from the SSM/T sounder on the DMSP satellites. The ultimate objectives of this research are to establish effective DI sounding algorithms for the microwave spectral region and to extend DI in directions which increase its utility as a practical sounding algorithm. DI has many attractive features in this application, including its close coupling to the physical formulation of the temperature sounding problem, its freedom from the necessity of using an *a priori* temperature profile, and its high level of computational efficiency. At present, we have understood the practical difficulties characteristic of temperature sounding in the microwave spectral region (as these apply to DI), and have produced software yielding temperature profiles of a convincing meteorological character. Research currently planned for the second year will continue with a detailed comparison with ground truth data. Theoretical advances in the past years' work include (in part), proper incorporation of window channel information in DI solutions for temperature profiles and the treatment of cross-track (nonzero nadir angle) scans. These theoretical advances will be included in the DI software, during the upcoming year, so as to further improve correspondence between DI temperature profiles and meteorological data.

*Electrical, Computer, and Systems Engineering Department, Boston University

1 Introduction

This document reports on the progress made in the first year of a research effort to apply the Differential Inversion (DI) algorithm for atmospheric temperature sounding to DMSP SSM/T-1 data. This algorithm was originally developed by Dr. Jean I. F. King of the Air Force's Phillips Laboratory [8] and has thus far been applied to infrared data collected with the TOVS/HIRS instrument [9]. The research carried out thus far has extended the theory of Differential Inversion in several directions necessary for successful application to microwave sounder data and the effort has progressed to the stage of initial applications of DI to DMSP SSM/T-1 microwave sounder data. The path for future developments leading to meteorologically useful temperature profiles from DI appears to be clear from the research carried out in the first year of this project.

In §2 of this report theoretical work on DI is discussed. Since DI is still considered to be a novel technique by the mathematical community addressing the treatment of inverse problems, a detailed theoretical exposition of DI is contained in §2.1. This treatment utilizes mathematical techniques derived from convolution algebras and recent work in hyperdistributions, which illustrates many important features in the foundations of DI which were difficult to understand in the original development from Laplace transform theory. A further discussion of hyperdistributions and convolution algebras is contained in Appendix A. A specific application of DI to the problem of atmospheric temperature sounding is contained in §2.2. Since DI is a general approach for the treatment of inverse problems, an application to problems in image restoration is included in §2.3. While not immediately relevant to problems in atmospheric temperature sounding, this DI-based image restoration technique may have applications for satellite remote sensing. We have found that it is important to include the contribution of the radiance from the ground correctly in the application of DI to microwave sounder data. In §2.4 the approach developed to include data from window channels is discussed. Lastly, in §2.5 the approach for application of DI to scans with nonzero nadir angle is discussed.

The work carried out thus far in applying DI to DMSP SSM/T-1 data is discussed in §3. In §3.1 the acquisition of DMSP data and the data path into our software is discussed. In §3.2 the DI software philosophy and implementation is described. It was considered particularly important that the software developed here which implements the DI algorithm be efficient for DMSP SSM/T-1 data, while being applicable in principle to a broad range of possible instruments. The fundamental character of the DI algorithm makes this a reasonable objective. DI requires differentiation of radiance data for its implementation via evaluation

of the Eddington-King formula. Ordinarily, differentiation of experimental data is to be avoided, due to the well-known difficulties which arise due to experimental noise. The specific requirements of numerical estimators of radiance derivatives in DI as well as algorithms which minimize the effects of experimental noise are treated in §3.3 and in Appendix C. Preliminary results in obtaining atmospheric temperature profiles from DMSP SSM/T-1 data are contained in §3.4 and future directions for further research work are given in §3.5.

The conclusions that can be reached from the research work carried out at this time are given in §4.

2 Differential Inversion

2.1 General Theory of Differential Inversion

In this section we develop the mathematical formalism of Differential Inversion (DI) as a general technique for the solution of convolution integral equations. As such DI has applicability far beyond the temperature sounding problem, e.g. in difficult image restoration problems and other classes of inverse problems. Our approach in this section is based upon hyperdistributions and convolution algebras, which is actually a more intuitive approach than the original derivations based on Laplace transforms. DI yields an expansion for the solution of an integral equation, $\rho(x)$, in successively higher derivatives of the known (observed) function $\phi(x)$. The representation of ρ in terms of derivatives of ϕ should not be confused with a Taylor expansion of ϕ. This representation, as we show is in fact the "dual" of the Taylor expansion and is in the form of a moment expansion analogous to Gauss' multipole expansion. In the following section we shall specialize our development of DI to the specific example of the temperature sounding problem.

We write our integral equation as the convolution of a source function ρ with a kernel function K, yielding an observed function ϕ,

$$\phi = K * \rho \tag{1}$$

with the standard notation for the convolution product

$$f * g = \int_{-\infty}^{\infty} f(x-y)g(y)\,dy \tag{2}$$

The differential inversion method gives the unknown function ρ as a series of derivatives of the known function ϕ,

$$\rho(x) = \sum_{k=0}^{\infty} \beta_k \nabla^k \phi(x) = \phi(x) + \sum_{k=1}^{\infty} \beta_k \nabla^k \phi(x) \tag{3}$$

Where $\beta_0 = 1$ is consequence of energy conservation, (i.e. normalization of the instrumental response kernel)

$$\int_{-\infty}^{+\infty} K(x)\,dx = 1. \tag{4}$$

We call equation (3) the differential inversion formula. This formula has been derived previously using methods of Laplace transform theory in the inversion of the equation of radiative transfer [8],[9] and also in a different context (astrometry) by Eddington [4]. Here we show an alternate derivation of the differential inversion formula in terms of a class of singular function, hyperdistributions, which are defined in appendix A. Any solution of a convolution integral equation can be written as a differential inversion formula. This occurs for example when Green's functions for homogeneous systems are used, as in Gauss' multipole expansion or even for the Taylor series expansion.

We begin by substituting equation (3) into equation (1)

$$\begin{aligned} \phi &= K * \left(\sum_{k=0}^{\infty} \beta_k \nabla^k \phi \right) \\ &= K * \left(\sum_{k=0}^{\infty} \beta_k \nabla^k \delta * \phi \right) \\ &= \left(K * \sum_{k=0}^{\infty} \beta_k \nabla^k \delta \right) * \phi \end{aligned} \tag{5}$$

Since $\phi = \delta * \phi$,

$$K * \sum_{k=0}^{\infty} \beta_k \nabla^k \delta = \delta \tag{6}$$

which shows that the sum $\sum_{k=0}^{\infty} \beta_k \nabla^k \delta$ is the "convolution inverse" of K. We denote this expansion by Inv$[K]$ and write

$$\text{Inv}[K(x)] = \sum_{k=0}^{\infty} \beta_k \nabla^k \delta(x) \tag{7}$$

We write K in the same form (in a hyperdistribution representation):

$$K(x) = \sum_{n=0}^{\infty} \alpha_n \nabla^n \delta(x) \tag{8}$$

Where we can determine the coefficients α_n from the moments of the kernel (see appendix A).

$$\alpha_k = \frac{(-1)^k}{k!} \int_{-\infty}^{\infty} x^k K(x)\, dx \tag{9}$$

It is now possible to relate the coefficients α_k and β_k by a "discrete convolution" product or Bochner-Martin algebra[1]. We begin by substituting the hyperdistribution representation of $K(x)$ into equation (6)

$$\left(\sum_{n=0}^{\infty} \alpha_n \nabla^n \delta(x)\right) * \left(\sum_{k=0}^{\infty} \beta_k \nabla^k \delta(x)\right) = \delta(x) \tag{10}$$

and observing that

$$\nabla^n \delta * \nabla^k \delta = \nabla^{n+k} \delta \tag{11}$$

We conclude that the algebra of sums of derivatives of δ is closed under the convolution product. Furthermore,

$$\begin{aligned} 1 &= \alpha_0 \beta_0 \\ 0 &= \alpha_0 \beta_1 + \alpha_1 \beta_0 \\ 0 &= \alpha_0 \beta_2 + \alpha_1 \beta_1 + \alpha_2 \beta_0 \\ &\vdots \end{aligned} \tag{12}$$

that is, in general

$$\sum_{k=0}^{n} \alpha_k \beta_{n-k} = 0 \qquad (n > 0) \tag{13}$$

The generic restrictions on the differential inversion solution are 1) the derivatives of ϕ must exist in the at the point of interest, as can be seen from equation (3), 2) the series in equation (3) must converge, 3) the kernel must have finite moments as in equation (9). The differential inversion representation of $\rho(x)$ is a moment expansion, i.e. it involves a "global description" of the convolution kernel. Differential inversion has the following advantages:

- The unknown function is evaluated at the same point as the known function; the deconvolution can be performed on a data stream which 1) allows for real-time deconvolution, 2) does not require that large data have to be artificially broken down, and 3) permits natural vector implementation.
- Exponential and Gaussian kernels are treated naturally by this method and are very common in physical systems

- The differential inversion coefficients exist if the kernel has finite moments; no additional assumption is made about the shape of the kernel during the implementation of the method.

Typically, convolution integral equations are treated by Fourier methods. If we define $\tilde{\Phi}(k) = \mathcal{F}\{\phi(x)\}$, and $\tilde{K}(k) = \mathcal{F}\{K(x)\}$, then by the Fourier convolution theorem,

$$\begin{aligned} \rho(x) &= \mathcal{F}^{-1}\{\tilde{\Phi}(k)/\tilde{K}(k)\} \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-ikt}\frac{\tilde{\Phi}(k)}{\tilde{K}(k)}\,dk \end{aligned} \tag{14}$$

Convergent Fourier integrals represent point functions; they cannot represent distributions or hyperdistributions. Several difficulties can arise when implementing numerical Fourier transform techniques that do not apply to differential inversion. These difficulties are discussed in standard numerical analysis books [12] and we present them below:

- The procedure must take a portion of the signal of finite length and deconvolve that segment, whereas data will sometimes need to be processed in a continuous stream in time.

- Large data sets are often artificially broken up into sections and each section is deconvolved separately, to accommodate memory and computational limits when implemented.

- The implementation often takes the duration of the response to be the same as the period of the data.

- The procedure can go wrong if the transform of the response function is zero at some point, so that we can not divide by the transform of the response at that point.

- Numerical Fourier techniques assume that the input signal is periodic, whereas real data is often not repetitive.

In particular, we note the usual observations made upon considering equation (14) that difficulties, both formal and numerical, arise when $\tilde{K}(k)$ approaches zero, or if the Fourier integral is not convergent. In such instances, difficulties arise with the convergence and uniqueness of the inverse Fourier transform. For example, if $K(x)$ is a Gaussian function,

$K(x) = e^{-x^2}$, the transform is $\tilde{K}(k) = e^{-k^2/4}$ and so the Fourier integral in equation (14) is highly divergent.

$$\rho(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-ikx} e^{k^2/4} \tilde{\Phi}(k)\, dk \tag{15}$$

The direct treatment of this kernel is often beyond the scope of Fourier transform techniques. On the other hand, since convolution products are the multiplicative operation of a closed algebra in the space of hyperdistributions, deconvolutions (the "convolution inverse") are uniquely defined in this approach as shown below.

We now show that the Taylor expansion is a differential inversion series (rather than the DI series being a Taylor expansion). We write the familiar Taylor expansion as

$$\phi(x + \Delta) = \sum_{k=0}^{\infty} \beta_k \nabla^k \phi(x), \quad \beta_k = \frac{\Delta^k}{k!} \tag{16}$$

we now identify the source density, ρ, by the formula

$$\rho(x) = e^{\Delta \cdot \nabla} \phi(x) = \phi(x + \Delta) \tag{17}$$

The Taylor series written in differential inversion form is

$$\rho(x) = \sum_{k=0}^{\infty} \beta_k \nabla^k \phi(x) \tag{18}$$

and the corresponding integral equation and kernel function are

$$\phi = K * \rho \tag{19}$$

$$K = e^{-\Delta \cdot \nabla} \delta(x) \tag{20}$$

note that x is in general an n-vector.

The familiar multipole expansion of Gauss is a form of differential inversion. In fact, the multipole expansion can be written as

$$\phi(x) = \sum_{k=0}^{\infty} \alpha_k \nabla^k G(x) \tag{21}$$

where

$$\alpha_k = \int x^{\otimes k} \rho(x)\, d^3x \tag{22}$$

(here $\otimes$ denotes tensor product or power) and G (the Green's function) is given by

$$G(x) = -\frac{1}{4\pi r} \tag{23}$$

where r is the magnitude of the 3-vector x. Equation (21) is clearly of the form (3) with coefficients equal to the (tensor) moments of the density ρ. Appendix B gives the details of the connection between the multipole expansion and differential inversion.

2.2 Application to the Temperature Sounding Problem

In this section, we derive an exact solution for the equation of radiative transfer in an infinite atmosphere. The exact solution is in the form of an explicit analytic formula for the Planck function, which is a simple, known fuction of the atmospheric temperature, as a linear combin- ation of the derivitives of the radiance emitted by the atmosphere, as it is measured by a satellite equipped with infrared or microwave detectors. We don't elaborate the advantages of an analytic formula beyond pointing out that the dependence of the atmospheric temperature profile on the physical properties of the atmospheric constituents through the weight functions in analytic form makes for considerable insight on the cause of variations in the temperature profile in the variations of (the moments of) the weight functions. The standard starting point of the analysis is an integral equation for the Planck function, B, in terms of the upwelling radiance, R, expressed in terms of pressure as a measure of height. This is the integral form of the monochromatic equation of radiative transfer as it is usually written in problems of atmospheric radiative transfer.

$$R(p') = \int_0^\infty B(p)\, W(p'/p)\, \frac{dp}{p} \tag{24}$$

We now modify this equation by converting to logarithmic pressure (height-like) variables by the transformation $z = -\ln p$ and $z' = -\ln p'$.

$$R(z') = \int_{-\infty}^{+\infty} B(z) W(z' - z)\, dz \tag{25}$$

We note that this puts the equation of radiative transfer in the form of a convolution integral, and thus renders the problem amenable to DI and the convolution algebraic techniques developed in the previous section. If we apply DI, we obtain the Eddington-King formula in terms of the height-like variable z (we drop the prime at this point),

$$B(z) = \lambda_0 R(z) + \sum_{k=1}^{\infty} \lambda_k \frac{d^k}{dz^k} R(z) \tag{26}$$

where the coefficients λ_k are the moments of the convolution inverse of the weight function W discussed below. Before giving the derivation of the Eddington-King formula it is worth summarising in words its content. The unknown temperature profile is contained in the Planck function in a simple way so that it is easy to calculate $T(z)$ given $B(z)$. The radiance R is known as a function of its argument so it and its derivitives are considered known. Finally, the coefficients λ_k will be shown below to be simply determined from the moments of the known weight W. Thus the Eddington-King formula very explicitly expresses the desired atmospheric temperature profile in terms of the available information, namely the upwelling radiance and atmospheric weight functions.

We now demonstrate that the solution of the integral equation (24) is given in the paper by *Hohlfeld and collaborators* [6]. The three proofs by Eddington, King, and Hohlfeld differ in significant ways and it is worthwhile to briefly remark on these differences. The Eddington proof is restricted to Gaussian weight functions (in fact it was given in very different context). As a consequence the formula given by Eddington for the coefficients of the radiance derivative are subject to this restriction which is quite untenable for the work we are carrying out. King used bilateral Laplace transforms to construct his coefficients and achieved a great generalization of the weight function forms. Finally, Hohlfeld's proof relies on the very general properties of convolution algebras and thus avoids difficulties with the stability of bilateral laplace transforms. Also, Hohlfeld's construction of the coefficients is given in terms of the Bochner-Martin algebra power series. This algebra can be implemented very efficiently on digital computers.

The Hohlfeld construction of the solution of the integral equation starts with the convolution form of the equation of radiative transfer expressed in terms of the height-like variable,

$$R = W * B \tag{27}$$

and constructs the solution by an appropriate construction of the convolution inverse of the weight function defined by

$$(Inv[W] * W)(x) = \delta(x) \tag{28}$$

where δ denotes the Dirac delta function.

This approach seems the most direct possible. In the paper by Hohlfeld and co-workers it is shown that singular functions ("hyperdistributions") of the form

$$\sum_{k=0}^{\infty} a_k \nabla^k \delta(x) \tag{29}$$

(here ∇ denotes the derivative which can be multi-dimensional) are closed under the operation of convolution, that is the convolution product of the two sums of the form (28), but in general with the different coefficients is also a sum of the form (28). Specifically,

$$(\sum_{k=0}^{\infty} a_k \nabla^k \delta(x)) * (\sum_{l=0}^{\infty} b_l \nabla^l \delta(x)) = \sum c_m \nabla^m \delta(x) \tag{30}$$

where the coefficients of the product hyperdistribution are given by the *finite* sum

$$c_m = \sum_{k=0}^{u} a_{n-k} b_k \tag{31}$$

Equation (30) defines the Bochner-Martin algebra. If the set of coefficients a and b are known, then (30) determines the coefficients c. Vice-versa, if the coefficients a and c are given, (30) determines the set of b's.

If the weight function of our equation (26) for the upwelling radiance is expanded into a hyperdistribution then (30) determines the coefficients of $Inv[W]$ which earlier enter in (27). We can then convolve both sides of (26) with $Inv[W]$. The result is

$$Inv[W] * R = B \tag{32}$$

If we now substitute the expression (28) for $Inv[W]$ into (31), we obtain exactly the Eddington-King formula (25).

The only step that remains to be completed is that of determining the coefficients in formula (28). Suppose the weight function W is expanded as a hyperdistribution, that is we consider the expansion

$$W(x) = \sum_{k=0}^{\infty} a_k \nabla^k \delta(x) \tag{33}$$

Taking successive moments of equation (32), we find

$$a_k = \frac{(-1)^k}{k!} \int_{-\infty}^{\infty} W(x) x^k dx \tag{34}$$

thus we have the simple result that, apart from the numerical coefficient, the coefficients of a hyperdistribution expansion *are the moments of the function itself.* The Hohlfeld derivation is thus complete. In light of this derivation, the Eddington-King formula can be interpreted as follows: The Planck function is the linear combination of the upwelling radiance derivatives whose coefficients are precisely $(-1)^n/n!$ times the moments of the convolution inverse of the atmospheric weight function. These moments are determined numerically by the efficient Bochner-Martin algebra contained in eq(30).

It is interesting to observe that the Eddington derivation proceeds from a Taylor expansion of the weight function and required considerable manipu- lation. The resulting formula (25), the Eddington-King formula, is not a Taylor expansion. However, because the two distinct functions B and R are involved rather then a single one as in the Taylor formula. Interestingly, the converse is true. Every Taylor formula is a special case of an Eddington-King formula. Applications of the Eddington-King formula are being developed in several different contexts (including biomedical). In fact, the very general procedure provided by the Hohlfeld derivation is applicable in the general context of inverse problems, an extremely promising area of research.

2.3 Two-Dimensional Differential Inversion with Separable Kernel for Image Restoration

In this section we apply DI to problems in image restoration. DI has already been applied to image restoration in intravascular ultrasonography. In that application DI is attractive because the point spread function of the instrument extends over many pixels and is poorly defined, loeading to inadequate performance of conventional image restoration techniques. We believe that this application of DI may be extended to other imaging problems, as in satellite remote sensing, and to other classes of inverse problems formulated in terms of convolution integral equations.

We write an observed image as the convolution of some (unknown) true image with an instrumental filter function (alternatively denoted as a point spread function), i.e.

$$I = F * G \tag{35}$$

with the defined quantities

$$I = I(\mathbf{X}) = \text{observed image} \tag{36}$$

$$F = F(\mathbf{X}) = \text{filter function} \tag{37}$$

$$G = G(\mathbf{X}) = \text{undegraded image} \tag{38}$$

and $\mathbf{X} = (x, y)$. Writing out the convolution integral explicitly,

$$I(\mathbf{X}) = \int G(\mathbf{X}')F(\mathbf{X}' - \mathbf{X})\, d\mathbf{X}'. \tag{39}$$

It is common in many problems arising in image restoration to assume that the filter function is separable. This arises when the instrument in question generates a point spread function which is a function of two independent coordinates in the output image. If we assume the filter function is separable, i.e.

$$F(\mathbf{X}' - \mathbf{X}) = F_x(x' - x)F_y(y' - y), \tag{40}$$

then we can now define a "reduced" filter function,

$$H(x', y) = \int F_y(y' - y)G(x', y')\, dy'. \tag{41}$$

We note that this reduced filter function is still a function of both image coordinates, x and y. Substituting the definition for $H(x', y)$ in our original form of the two-diminsional convolution integral yields

$$I(x, y) = \int F_x(x' - x)H(x', y)\, dx'. \tag{42}$$

This is now a convolution integral soluble by DI. Let $\lambda_0, \lambda_1, \lambda_2, \ldots$ denote the inversion coefficients for the kernel F_x. The resulting DI series is:

$$H(x', y) = \tag{43}$$

$$= \lambda_0 I(x, y)|_{x=x'} + \lambda_1 \frac{\partial I(x, y)}{\partial x}|_{x=x'} \tag{44}$$

$$+\lambda_2 \frac{\partial^2 I(x, y)}{\partial x^2}|_{x=x'} + \cdots \tag{45}$$

$$+\lambda_n \frac{\partial^n I(x, y)}{\partial x^n}|_{x=x'} + \cdots \tag{46}$$

Substituting into the definition of $H(x',y)$, we obtain

$$\int dy'\ F_y(y'-y)G(x',y') = \tag{47}$$

$$= \lambda_0 I(x,y)|_{x=x'} + \lambda_1 \frac{\partial I(x,y)}{\partial x}|_{x=x'} \tag{48}$$

$$+\lambda_2 \frac{\partial^2 I(x,y)}{\partial x^2}|_{x=x'} + \cdots \tag{49}$$

$$\cdots + \lambda_n \frac{\partial^n I(x,y)}{\partial x^n}|_{x=x'} + \cdots \tag{50}$$

Now the l.h.s. of this equation is a convolution integral which is also soluble by DI in its familiar one-dimensional formulation. Define $\mu_0, \mu_1, \mu_2, \ldots$ to be the inversion coefficients generated by the kernel F_y. We then obtain

$$G(x',y') = \tag{51}$$

$$= \mu_0 \left(\lambda_0 I(x,y) + \lambda_1 \frac{\partial I}{\partial x} + \lambda_2 \frac{\partial^2 I}{\partial x^2} + \cdots\right)|_{x=x'} \tag{52}$$

$$+\mu_1 \frac{\partial}{\partial y}\left(\lambda_0 I(x,y) + \lambda_1 \frac{\partial I}{\partial x} + \lambda_2 \frac{\partial^2 I}{\partial x^2} + \cdots\right)|_{x=x'} \tag{53}$$

$$+\mu_2 \frac{\partial^2}{\partial y^2}\left(\lambda_0 I(x,y) + \lambda_1 \frac{\partial I}{\partial x} + \lambda_2 \frac{\partial^2 I}{\partial x^2} + \cdots\right)|_{x=x'} \tag{54}$$

$$+\cdots \tag{55}$$

$$+\mu_n \frac{\partial^n}{\partial y^n}\left(\lambda_0 I(x,y) + \lambda_1 \frac{\partial I}{\partial x} + \lambda_2 \frac{\partial^2 I}{\partial x^2} + \cdots\right)|_{x=x'} \tag{56}$$

$$+\cdots \tag{57}$$

This expression is now readily interpreted as a product of two operator series equations:

$$G(x',y') = \tag{58}$$

$$= \left[\left(\sum_{i=0}^{\infty} \mu_i (\partial_y)^i\right)\left(\sum_{j=0}^{\infty} \lambda_j (\partial_x)^j\right)\right] I(x,y)\Big|_{X=X'} \tag{59}$$

$$= \left[\left(\sum_{i=0}^{\infty} \mu_i \partial_y^i\right)\left(\sum_{j=0}^{\infty} \lambda_j \partial_x^j\right)\right] I(x,y)\Big|_{X=X'} \tag{60}$$

The quantity in brackets may be viewed as a 2D DI operator acting on $I(X)$. In effect, we may interpret this as a *deconvolution operator* in 2D.

2.4 Inclusion of Window Channels in Differential Inversion

In its original formulation [8] [9] DI took no account of the presence of the ground as a radiating source. The atmosphere was presumed to be optically thick and all weight functions

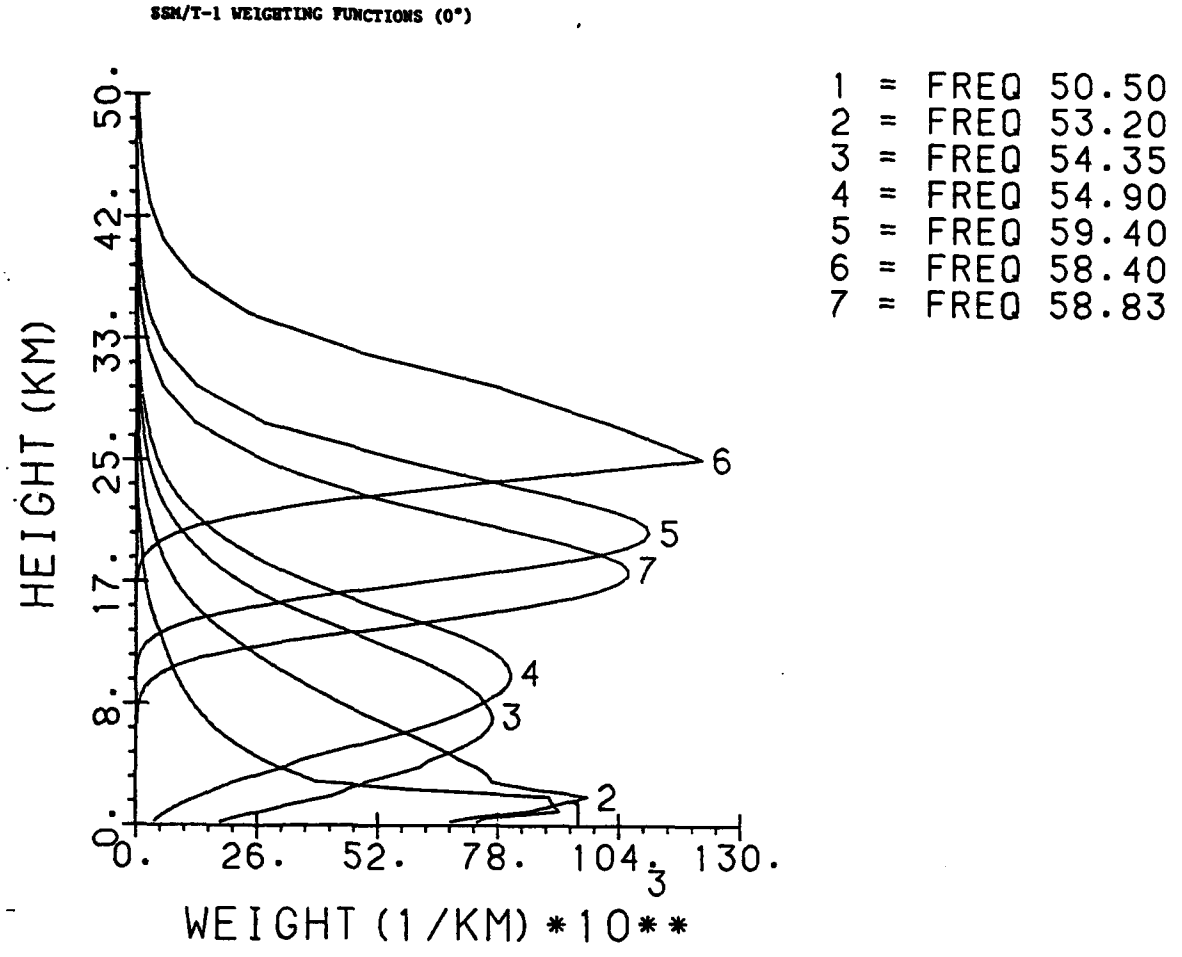

Figure 1: DMSP SSM/T-1 weight functions as a function of altitude in kilometers. Note the truncation at ground level of channels 1 through 4.

reached negligible values at altitudes well above the ground. We found that this limitation in the theory was not unduly limiting in applying DI to infrared radiances, such as for TOVS. In effect, the ground was treated as an infinitely deep, isothermal atmosphere grafted on below the actual atmosphere.

This approach is fundamentally inappropriate for application of DI to DMSP SSM/T radiances. In Figure 1, we show the SSM/T-1 weighting functions as a function of altitude. It is apparent that channels 1 through 4 have measurable contributions at ground level, and for channels 1 and 2, the area of the weight function extrapolated below ground level would be comparable to the area of the weight function above ground level. It is apparent, then, that we cannot treat the presence of the ground as an afterthought in applying DI in the microwave waveband.

We shall sketch a theoretical approach here which permits us to retain the desirable features of DI, in particular treatment of the weight function in the theory only in terms of its moments, while still dealing with this truncation of the weight function at ground level in an explicit fashion. It suffices to understand this approach to consider a few schematic sketches of the weight function in some limiting cases. We shall leave aside the problem of including the emissivity of the ground surface in this discussion, since our principle concern at this time is the truncation of the weight function. The essential features of this approach depend on the assumption of thermodynamic equilibrium, and on the linearity of the problem in terms of the weight function (see equations (24) and (25)).

If we were to imagine directing a radiometer towards a surface with no intervening atmosphere ("temperature sounding on the moon"), we would measure a brightness temperature corresponding to the physical temperature of the surface (emissivity presumed to be unity in this discussion). We expect this result for a problem described by thermodynamic equilibrium in which the surface emits a blackbody spectrum of radiation. In effect, the "atmospheric weight function" in this limit is simply a Dirac delta function, $\delta(z)$. (See Figure 2.) This form of the weight function is consistent with an application of DI, since the only nonzero moment of $\delta(z)$ is the zeroth moment, and so we obtain $\lambda_0 = 1$ and all the other λ's vanish. The Eddington-King formula then reduces to

$$B(z=0) = R(z=0) \tag{61}$$

as demanded by our original assumption of thermodynamic equilibrium.

We now return to our problem involving truncation of a realistic weight function at ground level. Linearity of the equation of radiative transfer with respect to the atmospheric weight function suggests that we include contributions both from the continuous piece of the weight function below the ground level, and a delta function contribution at the ground. The relative contributions of these two pieces should be computed on the basis that the total weight function should have unit area (i.e. be energy-conserving).

$$W(z) = W_a(z)H(z) + (1-A)\delta(z) \tag{62}$$

where

$$A = \int_0^\infty W_a(z)\, dz \tag{63}$$

with $W_a(z)$ the atmospheric contribution to the weight function, and $H(z)$ the Heavizide function. This choice of weight function is sketched in Figure 3.

A weight function of this form is easily accommodated within DI, since we may take moments of the composite weight function shown in equation (62) and then proceed to

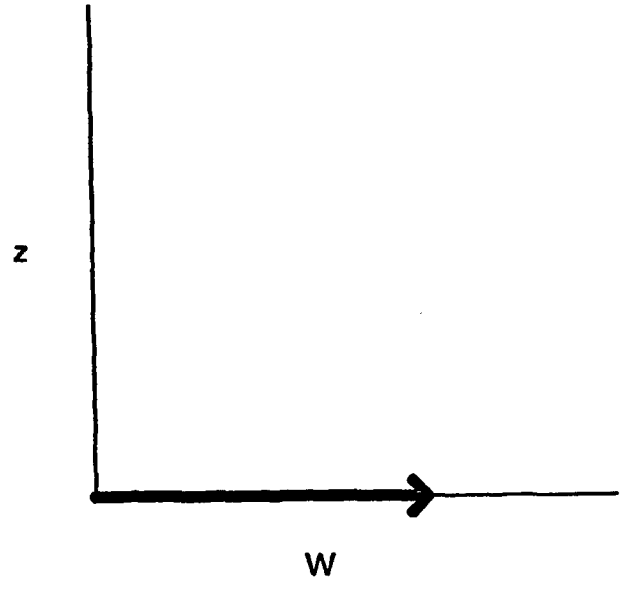

Figure 2: Weight function in the limit of zero atmospheric contribution, i.e. a "pure" window channel. The weight function tends to a Dirac delta function peaking at the ground.

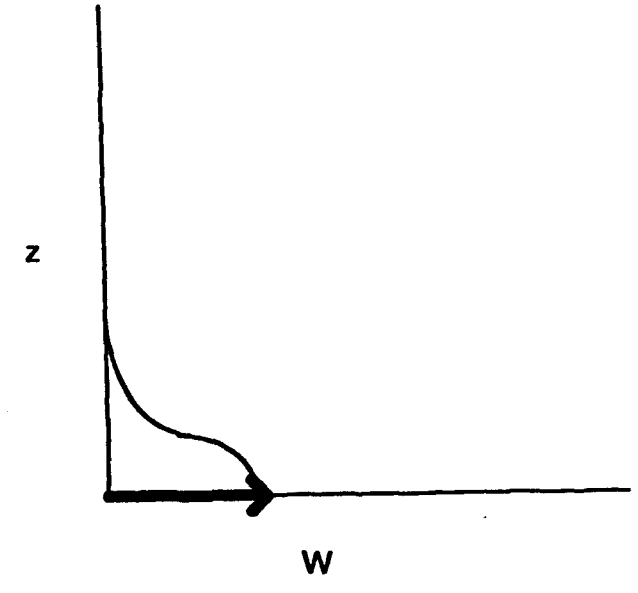

Figure 3: Weight function composed of a sum of a continuous contribution from the atmosphere and a singular (delta function) contribution at ground level.

calculate the λ's in the usual fashion. If we are constructing a moment expansion about an altitude $\bar{z}$, the nth moment of the weight function becomes

$$\int_0^\infty (z-\bar{z})^n W_z(z)\,dz + \int_{-\infty}^{+\infty} (z-\bar{z})^n (1-A)\delta(z)\,dz = \int_0^\infty (z-\bar{z})^n W_z(z)\,dz + (1-A)(-\bar{z})^n \tag{64}$$

where the second term represents the contribution of the ground component to the weight function to the moment of the weight function. In this approach, $\lambda_0 = 1$ identically and for each λ_i for $i \geq 1$, there are contributions from the ground due to each of the first i moments of the weight function. The effects of the ground are completely absorbed in the modification of λ values as a result of decomposition of the weight function into a contribution from the atmosphere and a singular contribution from the ground.

In this approach we take the value of $\bar{z}$ to be an altitude which is associated with each channel. In the previous formulations of DI, this has often been taken to be the altitude of the maximum value of the weight function, although a detailed theoretical justification for that choice was never developed. We now see in the convolution algebra derivation of DI that $\bar{z}$ is interpreted as the altitude about which we conduct the moment expansion of the weight function. Thus the choice of $\bar{z}$ is in some sense arbitrary, though a choice of $\bar{z}$ close to the mean value of the weight function yields a set of λ's which go to zero as rapidly as possible, which helps ensure rapid convergence of the Edddington-King formula.

This prescription for the selection of $\bar{z}$ is consistent with the decomposition of the weight function into nonsingular atmospheric components and a delta function contribution at ground level. The mean of the composite weight function is well defined (since the first moment is well defined), and so $\bar{z}$ can be determined. As the singular contribution to the weight function becomes more and more important (when the weight function becomes more severely truncated), the value of $\bar{z}$ tends a limit to the altitude of the ground. This choice of $\bar{z}$ is also consistent with the choice of locations for evaluation of the radiance derivatives.

Data from a window channel may easily be incorporated in a DI calculation if we model the window channel as a delta function which is nonzero at ground level. The value of $\bar{z}$ for such a channel is simply the altitude of the ground (which is then used directly in calculation of the radiance derivatives). The only nonzero inversion coefficient for the ground channel is $\lambda_0 = 1$. The measured radiance in that channel (corresponding to ground temperature in thermal equilibrium) is used in the same fashion as the other radiance channels used in the DI temperature profile calculation.

2.5 Temperature Profiles from Off-Nadir Scan Angles

The DMSP instrument is mounted on a scan platform which scans in the cross-track direction. To date, we have restricted attention to scans in which the radiometer is oriented directly downward, i.e. at a nadir scan angle.

The weight functions for these off-nadir scans are very asymmetric, with a shallow "shoulder" at high altitudes, a steep rise to a maximum value, and then a very steep drop to negligible amplitude. DI permits us to process these weight functions in exactly the same fashion as we have developed for the nadir scan angle radiances. Since DI depends only on the moments of the weight function, and these off-nadir scan angle weight functions still have well behaved moments, we can obtain inversion coefficients (λ's) appropriate to these weight functions and implement the Eddington-King formula as before. The weight functions will be different for each distinct scan angle, and so a distinct set of inversion coefficients will be required at each distinct scan angle.

We plan on working at Phillips Laboratory over the next several months computing a set of weight functions for each of the distinct scan angles of the SSM/T instrument. We will then derive the inversion coefficients and attempt to calculate atmospheric temperature profiles at all the scan angles of the SSM/T instrument. Our existing software will then be extended to process temperature profiles at each scan angle of the SSM/T instrument using the appropriate inversion coefficients.

3 Analysis of DMSP SSM/T Data by Differential Inversion

3.1 DMSP Data Acquisition and Data Path

The fundamental objective of this research effort is to demonstrate the feasibility of DI for analysis of DMSP SSM/T data to obtain meteorologically useful atmospheric temperature profiles. For these purposes a comparatively small number of scans is required to develop and test the DI algorithm as applied in the microwave band. Later work (to be pursued in the second half of this research effort) will be concerned with evaluating the performance of DI in comparison with ground truth measurements and in comparison with competitive sounding algorithms.

We obtain data from research workers at Phillips Laboratory who are engaged in other research with the DMSP SSM/T sounder. Primarily data has been acquired through Atmospheric and Environmental Research, Inc. (AER), which is working under contract with the United States Air Force. AER has the responsibility of taking data from the ground station at Phillips Laboratory (Hanscom AFB) and processing it in various ways. Researchers in AER agreed to provide us data in a simple ASCII format on a daily basis (as needed) generated by the DMSP overpass of the United States East Coast. This data translation avoids issues of our dealing with proprietary file formats and provides sufficient data for our needs at this time.

Data are provided as brightness temperatures ordered as to time and nadir scan angle. At present no status information is provided as to whether scans are valid, though most invalid scans are reasonably self-evident as wildly discrepant values of brightness temperature. At present we have been unable to obtain any information regarding error bars of the brightness temperatures, which would be an extremely useful information (even if only approximate error bars are available) in implementing DI on this data set. DI requires radiances rather than brightness temperatures, and so we convert brightness temperatures for channels into radiances using the Planck radiation formula.

Data files formatted for our use are provided on a SUN workstation at Phillips Laboratory and are transferred to our computers at Boston University using ftp (file transfer protocol) on Internet. We have determined that this procedure has a minimal impact on the work going on at Phillips Laboratory.

Throughout the period of time beginning in Spring of 1993, the ground station at Phillips Laboratory has had hardware problems which have invalidated most of the DMSP scans in our data sets (more than 70%). We emphasize that these problems have not prevented us from completing our own primary work of developing DI algorithms in application to DMSP SSM/T data since the number of scans we require at this time is fairly modest.

3.2 Implementation of Differential Inversion Software

We have begun the development of a program to demonstrate the application of DI to determination of atmospheric temperature profiles from DMSP SSM/T data. We have written this software in C owing to its portability to a variety of computational platforms and its universality in the academic environment. We feel that this is an appropriate choice at this time, since the objective is to develop the algorithm, rather than the production of operational code, per se. A future operational implementation can be produced in another programming language (e.g. Ada) if necessary.

DI is notable as an inversion technique in that the instrumental characteristics enter the theory in a particularly well-defined way, and thus we have an opportunity to develop a general piece of software for solution of the inverse problem which will be applicable to a wide variety of instruments. The properties of the instrument enter the DI calculation of the atmospheric temperature profile only through the values of the channel frequencies and through the values of the inversion coefficients (which also contain information about the atmosphere as well as the instrument). We can provide the channel frequencies to our DI program in a data file and either provide a file of pre-determined inversion coefficients or provide the weight functions and compute the inversion coefficients from the moments of the weight functions by the Bochner-Martin algebra.

It is worth noting here, in passing, that while DI has a significant computational "overhead", the computation of the inversion coefficients, the implementation of the Eddington-King formula is computationally straightforward and efficient. Therefore, DI is attractive as an inversion algorithm in treating very large data sets as are generated by current and planned generations of atmospheric temperature sounding instruments.

Since instrumental characteristics can be incorporated in the DI program in a straightforward way, we have elected to try to produce a general-purpose atmospheric temperature sounding program which can be applied to a variety of instruments operating in the microwave and infrared. The program as presently implemented can be applied to TOVS/HIRS, DMSP SSM/T, and EOS/AIRS (synthetic) radiance data. Instrumental dependencies are

incorporated solely through the specification of channel frequencies and corresponding weight functions. The operation and structure of the program is further discussed in Appendix D.

3.3 Differentiation of Radiance Data

DI involves the computation of the Planck function profile in the atmosphere using the logarithmic pressure derivatives of the upwelling radiance, weighted by the inversion coefficients. It is apparent that DI requires the determination of high quality radiance derivatives for its effective implementation. Numerical differentiation of noisy experimental data is usually to be avoided as a computationally difficult problem owing to the tendency for the noise to be "amplified" by the process of differentiation.

The Eddington-King formula for DI places an additional requirement on the numerical derivatives of the radiance data; all estimates of radiance derivatives must be of the same order in truncation error expressed as a stepsize of the sampling interval in z. For example, if we take the Eddington-King formula out to the fifth order, i.e. compute out to the $\lambda_5 \; d^5R/dz^5$ term, the conventional divided difference formula for d^5R/dz^5 will be $O(h^6)$, with h the interval in z between radiance channels. However, if we combine this term with the usual estimate of the dR/dz term given by a low order divided difference formula (which is $O(h^2)$), the truncation error will only be effectively of the order of the lowest order radiance derivative estimate, i.e. $O(h^2)$. Therefore, it is apparent that estimates of radiance derivatives must be calculated to a uniform, high order for all terms in the Eddington-King formula. The order of the radiance derivative estimates must conventionally be of one order higher than the order of terms retained in the Eddington-King formula.

An additional complication arises due to the fact that the z values of the channels of the DMSP SSM/T radiometer (and, indeed, of any practical sounding instrument) are unevenly spaced. Deriving high order divided difference formulae on unevenly spaced data by Taylor series calculations is extremely tedious and prone to error. We have found a Vandermonde matrix technique [7] to be highly effective in deriving such formulae (see Appendix C). Estimators of numerical derivatives obtained by such techniques are shown to be optimal. Furthermore, the derivative estimators are continuous functions which allows us to represent the DI temperature profiles as continuous curves rather than as discrete sets of points at which divided difference formulae are evaluated. (Compare with our earlier work in [9].)

The Vandermonde matrix technique also illuminates the relationship between the problem of derivative estimation and the problem of moments. As a consequence, it can be shown that when we are treating a small set of data points which we are considering simultaneously

(as is the case in our present treatment of DMSP SSM/T data) the estimation of numerical derivatives by Vandermonde methods is equivalent to fitting the data set with a polynomic curve and employing that curve to evaluate the radiance derivatives. This is the approach we use on the DMSP SSM/T data at present. Application of DI to an instrument with a larger number of channels (in which not all channels would be used to evaluate radiance derivatives at all heights in the atmosphere) would benefit greatly from the full application of the Vandermonde matrix techniques.

3.4 Preliminary Results

We have succeeded in applying the DI algorithm to the calculation of atmospheric temperature profiles from DMSP SSM/T data. The results obtained thus far are encouraging, though it is clear that, as the algorithm becomes progressively more developed, there is considerable room for improvement of the algorithm in the direction of production of more meteorological temperature profiles.

The first version of the DI program applied to DMSP SSM/T assumed the weight functions were of generalized exponential form [9],

$$W(p/\bar{p}) = \frac{m^m}{\Gamma(m+1)} (p/\bar{p}) \exp[-m(p/\bar{p})^{1/m}] \tag{65}$$

where m is a parameter (typically between 0.0 and 4.0), $\bar{p}$ is the pressure at which the weight function achieves a maximum, and p is the pressure varying through the atmosphere. Weight functions of this form had a great deal of utility in our earlier work obtaining atmospheric temperature profiles from infrared data gathered by the TOVS/HIRS instrument.

As a first step towards a functioning program, we assumed that all SSM/T channels were described by the same weight function (generalized exponential weight function with $m = 1.0$) and did not take into account any influence of the ground temperature or the truncation of weight functions at ground level. This was a useful first step in the implementation process since it essentially replicated our earlier results in the infrared, while showing the differences in the problem intrinsic to operating in the microwave band.

Typical results from this first level of development are given in Figures 4 through 7. These curves are very smooth, as may be expected from the simplistic level of the implementation just described. The curves are generated as a result of a single polynomic fit to the radiance through the atmosphere and no account is taken of the variation of weight function morphology through the atmosphere or variation in frequency for the different SSM/T channels.

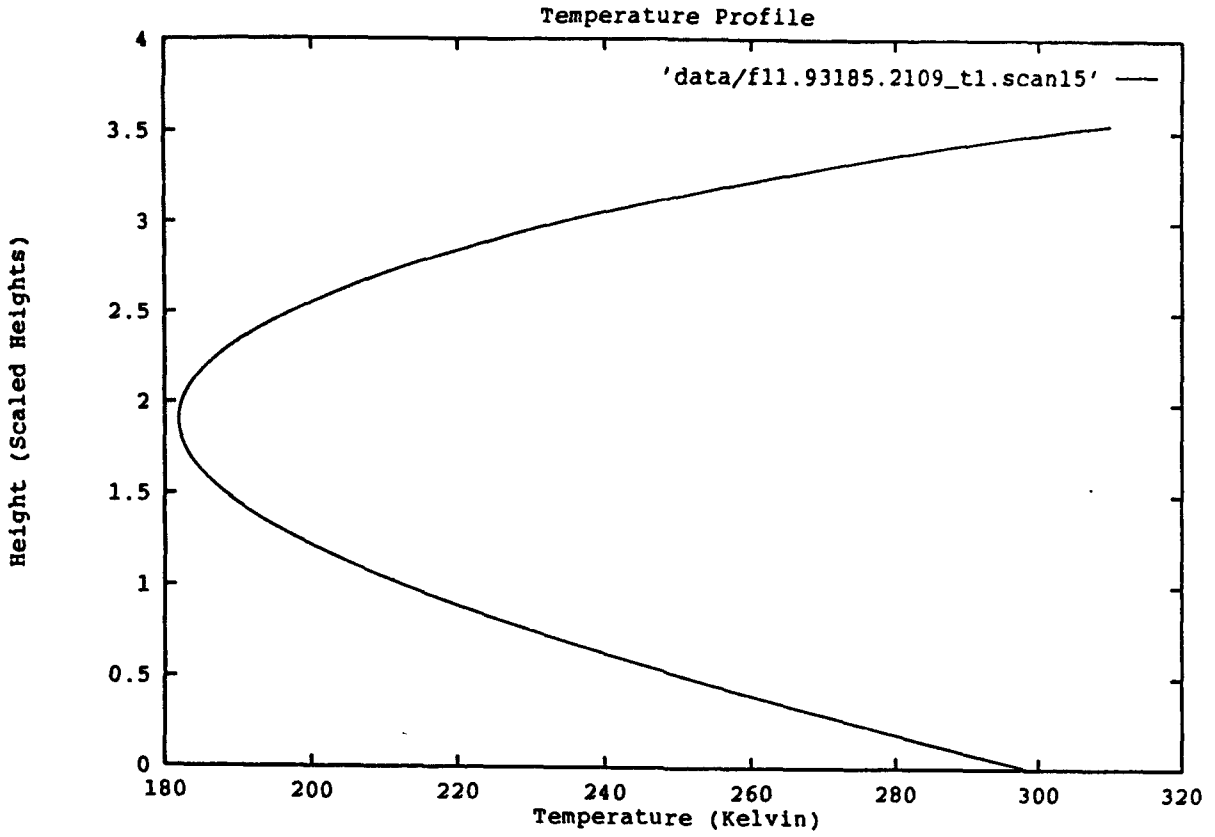

Figure 4: Atmospheric temperature profile obtained from the first version of the DI algorithm applied to DMSP SSM/T-1 data. The scale height is taken to be 8.45 kilometers. The curve is very smooth owing to the simple initial implementation of the algorithm and the fact that ground temperature information was not included. The general features of the temperature profile were as expected for this implementation and gave confidence that the algorithm was in an appropriate state of development for the next stages of development.

The profiles could (very loosely) be interpreted as having a tropopause at a plausible height, though the variation in temperature through the atmosphere is extreme.

It was apparent from these initial results that we would need to begin to extend DI from the version as implemented for the infrared band in our earlier work. We have already noted that the frequency spread of the DMSP SSM/T-1 channels cannot be neglected. Since the Eddington-King formula technically applies to monochromatic radiances and the Planck function at a single frequency, it is appropriate to identify a frequency with a given height in the atmosphere at which the Eddington-King formula is being evaluated. (Note that this approach is consistent with our practice of evaluation of continuous radiance derivative estimators by use of Vandermonde techniques.) Therefore, we use a continuous fit to the

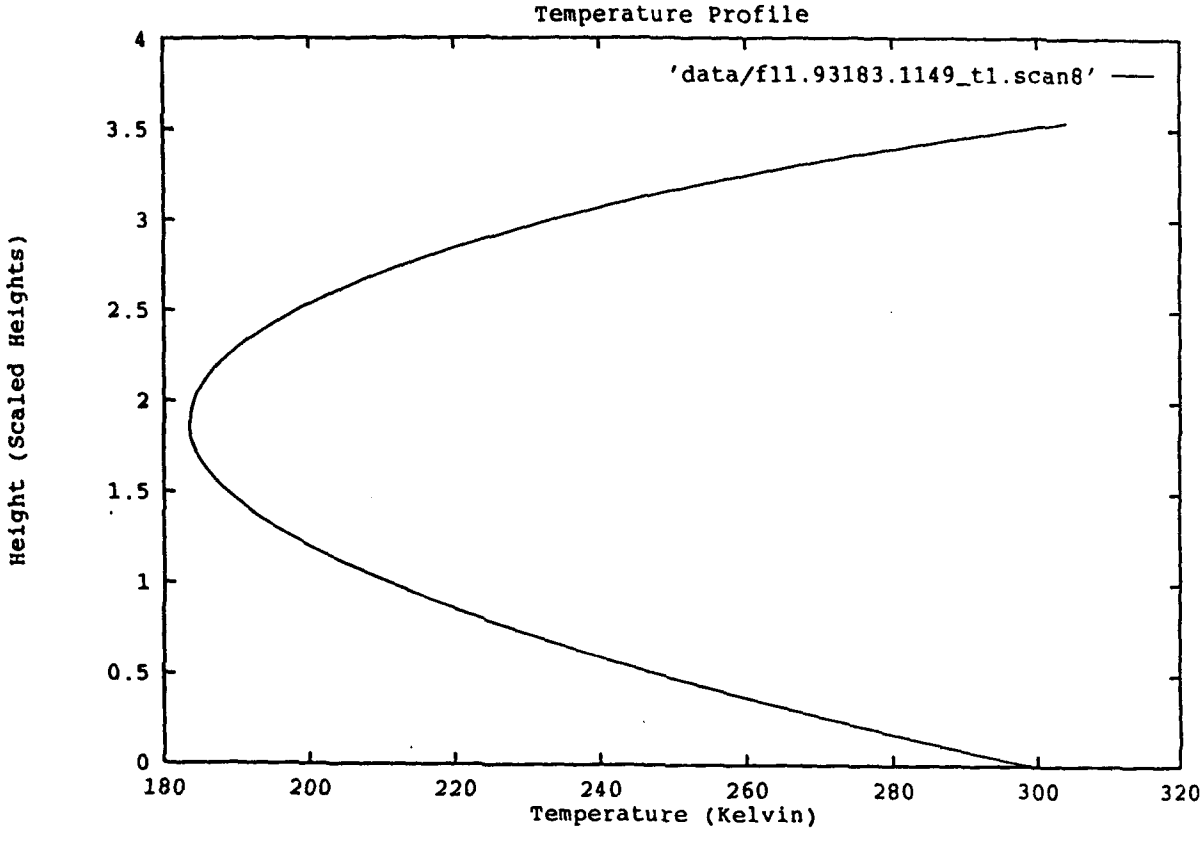

Figure 5: Typical atmospheric temperature profile obtained from the first version of the DI algorithm applied to DMSP SSM/T-1 data. For further information see the caption to Figure 4.

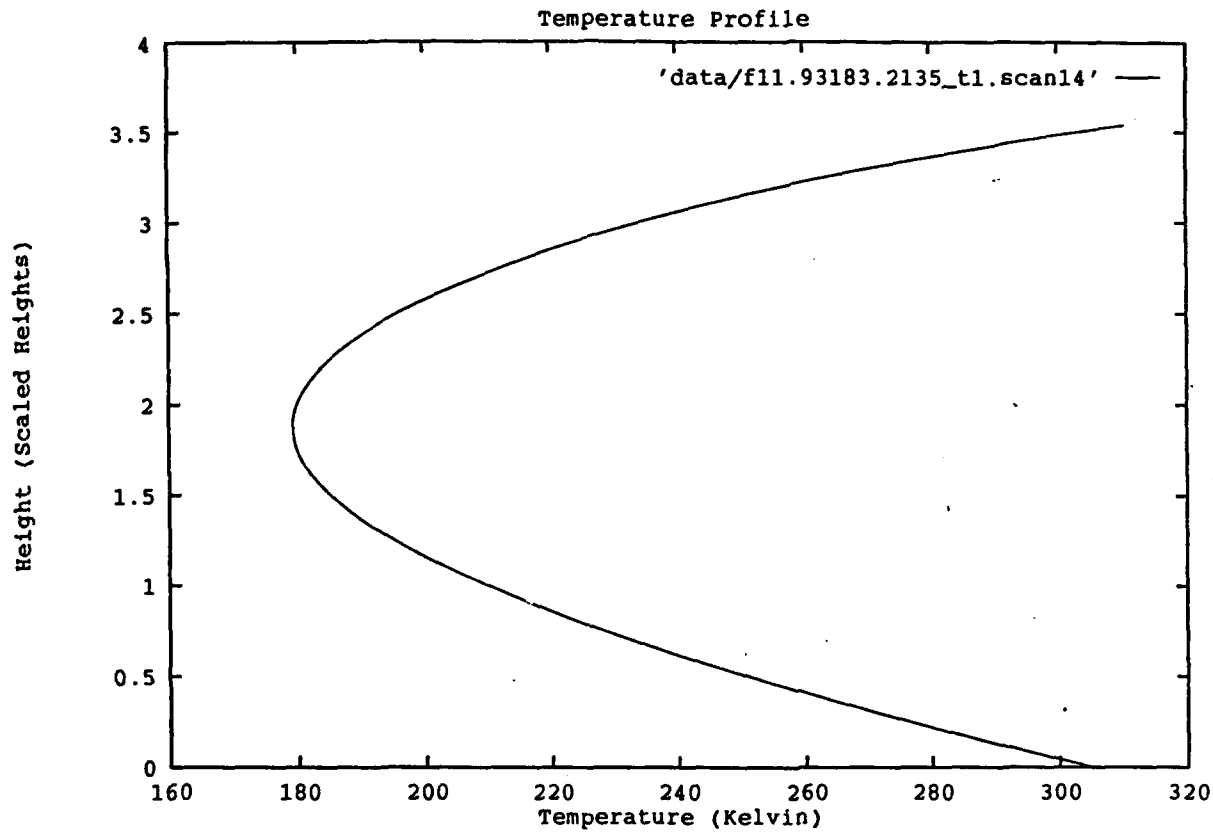

Figure 6: Typical atmospheric temperature profile obtained from the first version of the DI algorithm applied to DMSP SSM/T-1 data. For further information see the caption to Figure 4.

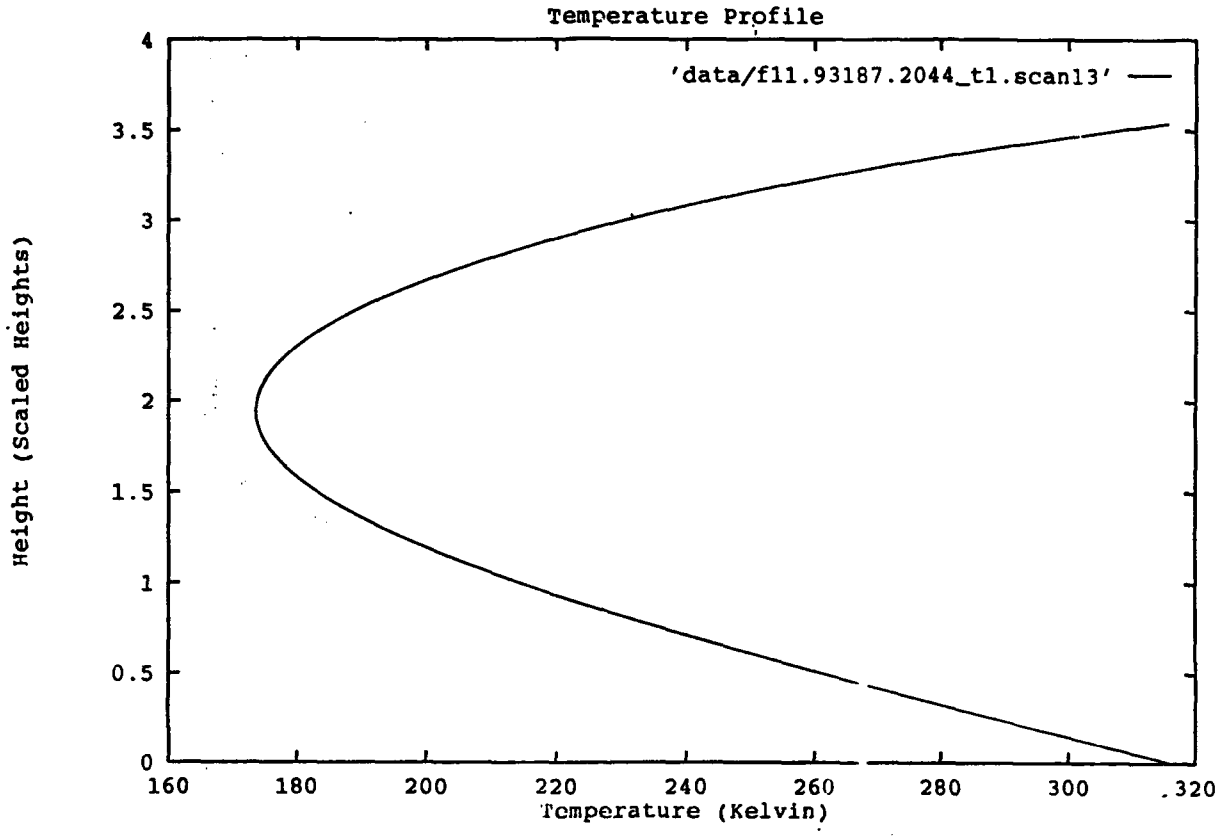

Figure 7: Typical atmospheric temperature profile obtained from the first version of the DI algorithm applied to DMSP SSM/T-1 data. For further information see the caption to Figure 4.

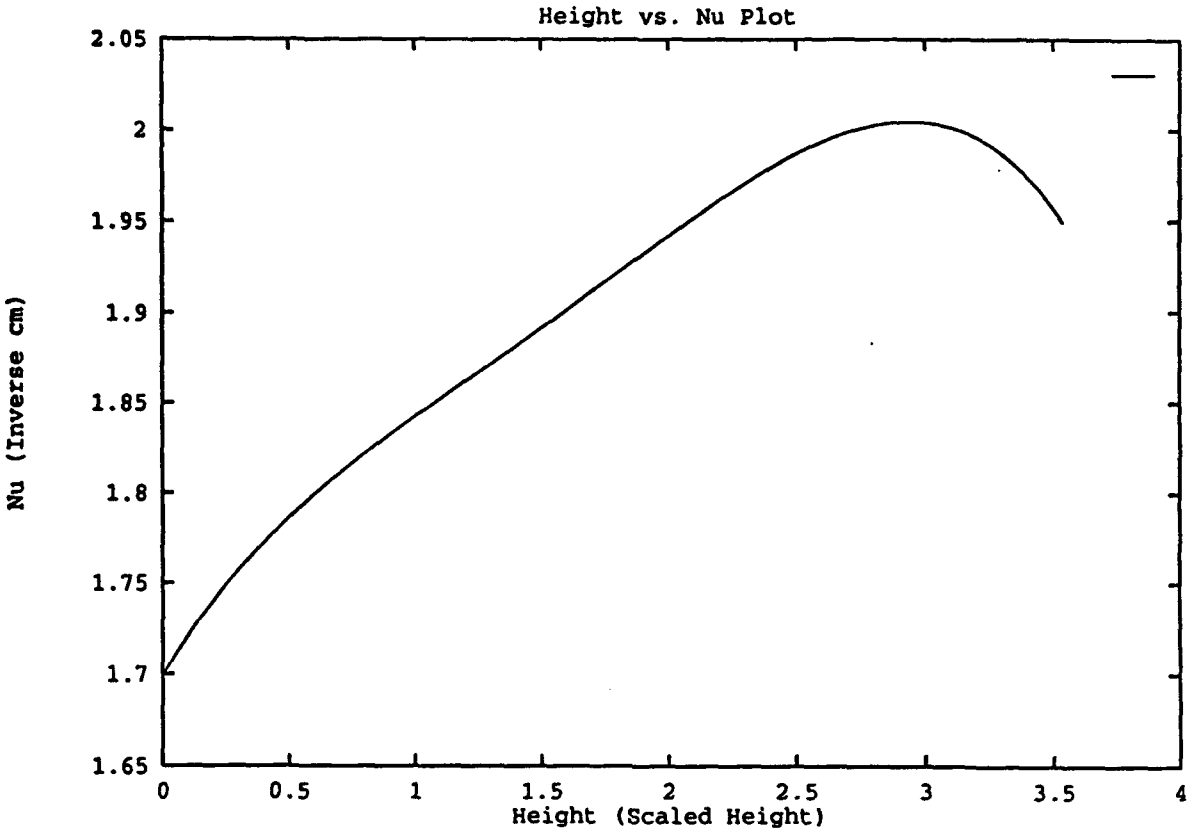

Figure 8: Fit to frequency versus altitude based on the frequency and $\bar{z}$ values of the DMSP SSM/T-1 instrument. This fit is used in the implementation of DI in terms of a monochromatic formula applied at a continuous range of altitudes.

frequency as a function of altitude, based on the frequencies and $\bar{z}$ values of the DMSP SSM/T-1 weight functions. This fit is shown in Figure 8.

We have also noted that the King weight functions do not match the actual weight functions of the DMSP SSM/T-1 instrument very well. This is unlike our previous experience in the infrared, where we found the King weight functions to be an acceptable fit to the weight functions of the TOVS/HIRS instrument. We have therefore used the DMSP SSM/T-1 weight functions to directly calculate moments and evaluate the λ coefficients. These λ coefficients are given out to λ_9 in the table below.

Chan. λ	1	2	3	4	5	6	7
0	0.99978	1.00015	1.00416	0.98433	0.95241	0.88932	0.96944
1	-0.06868	-0.39045	-0.21500	-0.07592	-0.47879	-0.83451	0.72259
2	-0.07954	-0.11438	-0.17211	-0.18652	0.00147	0.25109	0.14158
3	-0.02042	0.02224	0.01230	-0.00682	0.02531	0.00187	-0.04499
4	-0.00300	0.01265	0.01835	0.01756	-0.00055	-0.02306	-0.02944
5	0.00072	0.00290	0.00196	0.00308	-0.00019	0.01104	-0.00499
6	0.00073	-0.00023	-0.00096	-0.00065	-0.00008	-0.00418	0.00085
7	0.00029	-0.00041	-0.00035	-0.00031	0.00000	0.00118	0.00062
8	0.00006	-0.00016	-0.00003	-0.00004	0.00000	-0.00019	0.00013
9	0.00000	-0.00002	0.00001	0.00000	0.00000	-0.00004	0.00000

It can be noted from the table that the morphology of the weight functions changes significantly between the weight functions with $\bar{z}$ low in the atmosphere, compared to the weight functions with $\bar{z}$ at high altitudes. This strongly suggests that we should also allow the λ's to vary as a function of height to reflect this height-dependent change in weight function morphology. As discussed in the previous section, a window channel has $\lambda_0 = 1$ and all other λ's zero. This behavior is reasonable as a limit as $\bar{z} \rightarrow 0$ for the λ coefficients of the other channels. The behavior of λ_0, λ_1, and λ_2 as a function of altitude is given in Figures 9, 10, and 11 respectively. At present we indicate the λ values at the $\bar{z}$ values of the corresponding channels and join these points with straight lines. We believe, however, that it is appropriate to use a continuous fit to the shape of the λ curves with altitude. In effect, we would then be using a "virtual weight function", sampled in $\bar{z}$ space by the DMSP SSM/T-1 weight functions. As $\bar{z}$ is varied, this "virtual weight function" smoothly varies between the shapes of the DMSP SSM/T-1 weight functions. This is also consistent with the use of the Eddington-King formula as a monochromatic inversion formula.

We also incorporate in our implementation of DI at this point a "pure" window channel with $\lambda_0 = 1$ and all other λ's zero, as discussed in §2.4. Such a window channel is included in the representation of the values of the inversion coefficients as a function of channel height, shown in Figures 9 through 11. The intention here is to use an independent determination of temperature at ground level to compute a blackbody radiance along with the radiances measured by the DMSP SSM/T-1 instrument to obtain a temperature profile with an improved match to the actual temperature profile, particularly at low altitudes. Such an independent determination of the temperature at ground level could come either from a ground station or from another instrument. We intend to investigate the suitability of the ground temperature

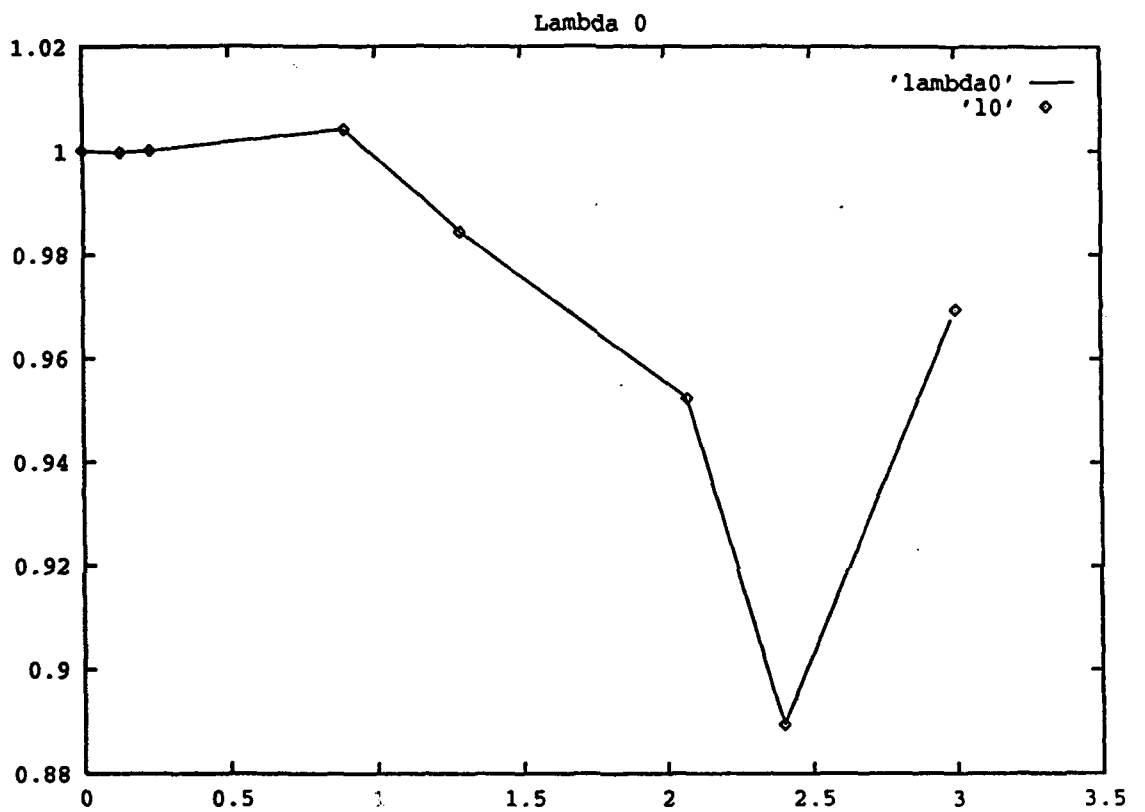

Figure 9: Values of λ_0 as a function of $\bar{z}$ for the channels of the DMSP SSM/T-1 instrument. The value of $\lambda_0 = 1$ at $\bar{z} = 0$ corresponds to a pure window channel added to the usual set of DMSP SSM/T-1 channels. Note that the λ_0 value of this window channel is consistent with the run of λ_0 as a function of $\bar{z}$ for the other DMSP SSM/T-1 channels.

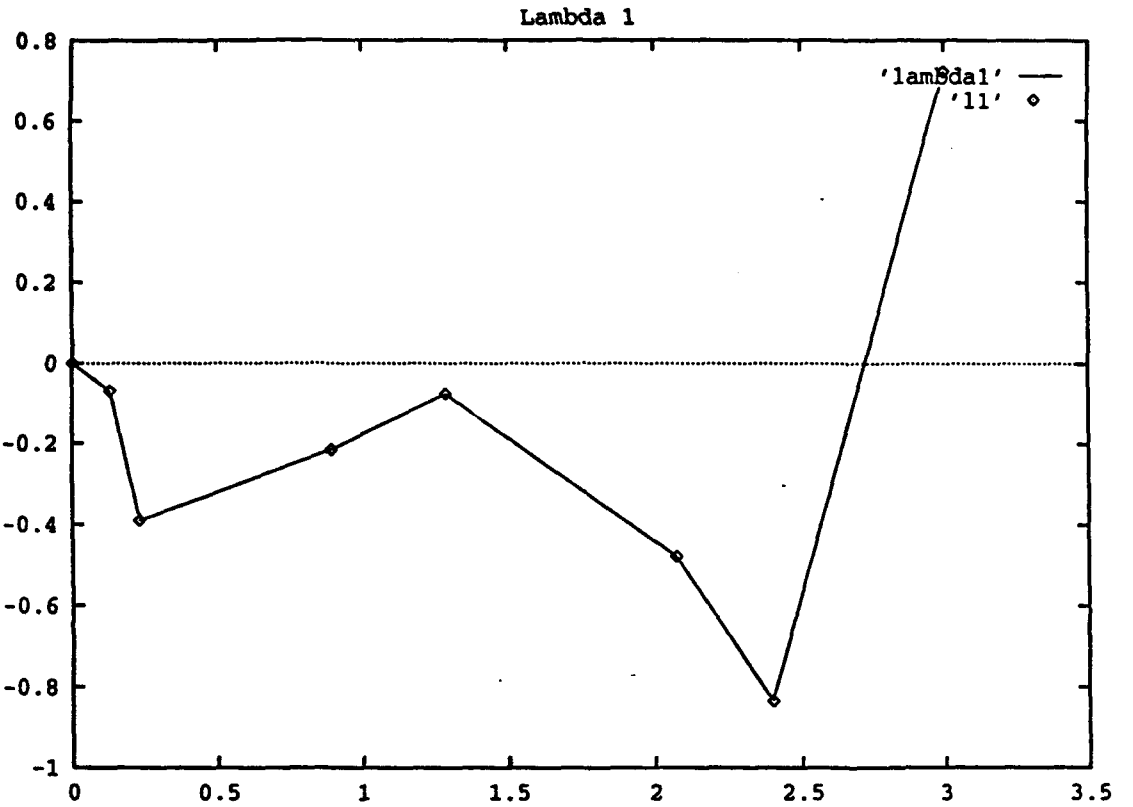

Figure 10: Values of λ_1 as a function of $\bar{z}$ for the channels of the DMSP SSM/T-1 instrument. The value of $\lambda_1 = 2$ at $\bar{z} = 0$ corresponds to a pure window channel added to the usual set of DMSP SSM/T-1 channels. Note that the λ_1 value of this window channel is consistent with the run of λ_1 as a function of $\bar{z}$ for the other DMSP SSM/T-1 channels.

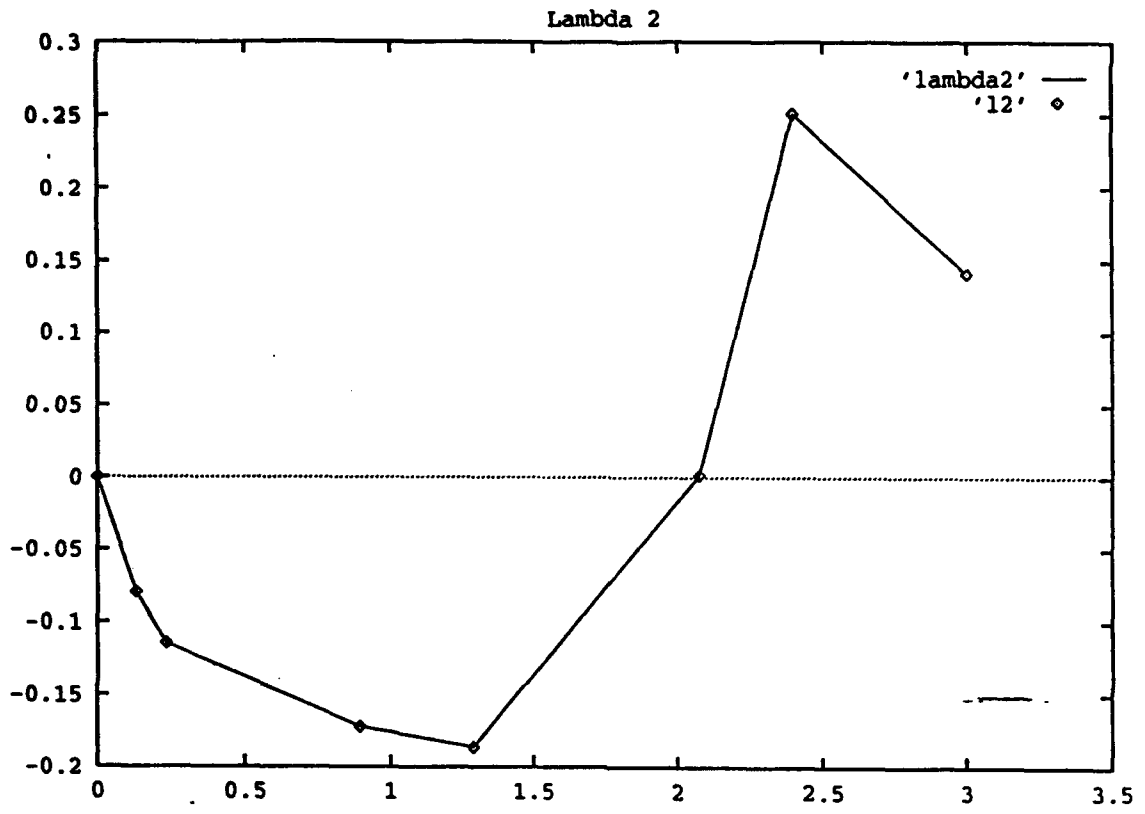

Figure 11: Values of λ_2 as a function of $\bar{z}$ for the channels of the DMSP SSM/T-1 instrument. The value of $\lambda_2 = 1$ at $\bar{z} = 0$ corresponds to a pure window channel added to the usual set of DMSP SSM/T-1 channels. Note that the λ_2 value of this window channel is consistent with the run of λ_2 as a function of $\bar{z}$ for the other DMSP SSM/T-1 channels. The behavior of the higher order inversion coefficients is also consistent with the behavior shown here for λ_1 and λ_2.

obtained from the DMSP SSM/I instrument [10] to generate such temperature values. The operational advantages of combining data from two instruments on the same satellite should be considerable.

We have calculated DI temperature profiles incorporating frequency variation with height, λ coefficient variation with height, and a value of the ground temperature applied to a pure window channel. We have assumed, for demonstration purposes only, a constant temperature of 285 °K for the ground. We expect future work will benefit from the use of an actual determination of ground temperature either from ground station observations, or from observations from another instrument. Typical DI temperature profiles determined incorporating these new extensions are show in Figures 12 through 15.

These improved temperature profiles exhibit more reasonable characteristics than those shown in Figures 4 through 7. The sharp discontinuities in slope of the temperature profile are doubtless due to our use of a piecewise linear fit to the behavior of the inversion coefficients (λ's) as a function of height, since the values of $T(z)$ obtained depend in an important way on the weight functions through the inversion coefficients. We expect continued algorithmic development to continue to yield improvements in the determination of the atmospheric temperature profile.

3.5 Plans for Future Work in DMSP SSM/T Data Analysis

On the basis of the progress made in this research effort to date, we can identify areas for further research which will represent the largest opportunities for advances in the production of meteorologically useful temperature profiles using DI applied to DMSP SSM/T-1 data. These areas for further research are outlined briefly here:

- A fit to the inversion coefficients as a continuous function of height will be obtained. This will replace the piecewise linear fits used in the present version of the DI software. It is expected that this will eliminate the slope discontinuities which are the most striking non-physical feature of the temperature profiles at present.

- It is important to calculate new weighting functions for the DMSP SSM/T-1 instrument, or to obtain weight functions in a digitized form from previous computations. The weight functions used presently were digitized from plots kindly provided by Mr. Falcone of Phillips Laboratory. However, this source is not sufficiently accurate to

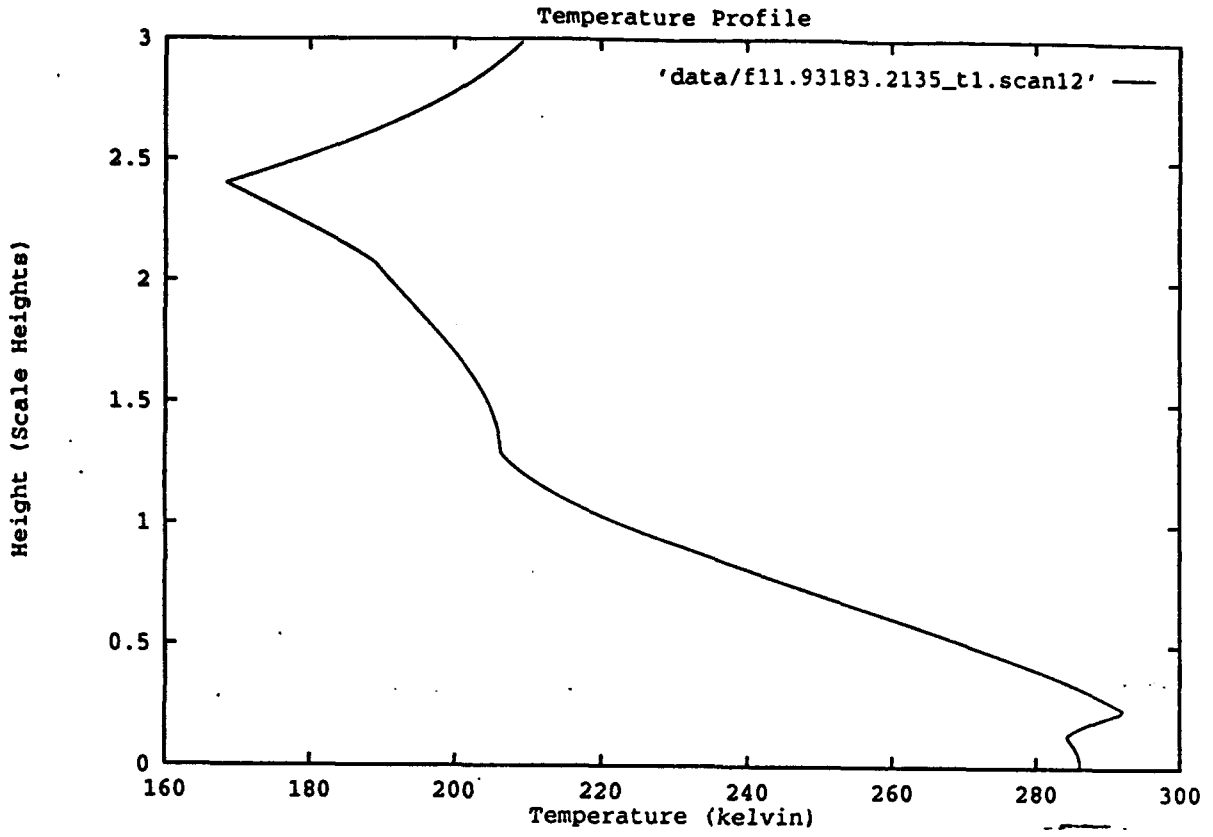

Figure 12: Atmospheric temperature profile obtained from the present version of the DI algorithm applied to DMSP SSM/T-1 data. The scale height is taken to be 8.45 kilometers. The algorithm has been extended to include variation of channel frequency with height, variation of the shape of weight functions with height, and the radiation from the ground. Here, for demonstration purposes, a ground temperature of 285 °K is chosen; in actual practice a temperature from a ground station or measured with a separate instrument with a "pure" window channel would be used. The character of the temperature profiles is considerably improved compared to the first implementation of DI, as contained in Figures 4 through 7. The slope discontinuities in these profiles arise from the piecewise-linear fit to the shape variations of the weight functions with height and are not expected to be present in temeperature profiles generated by the next implementation of DI.

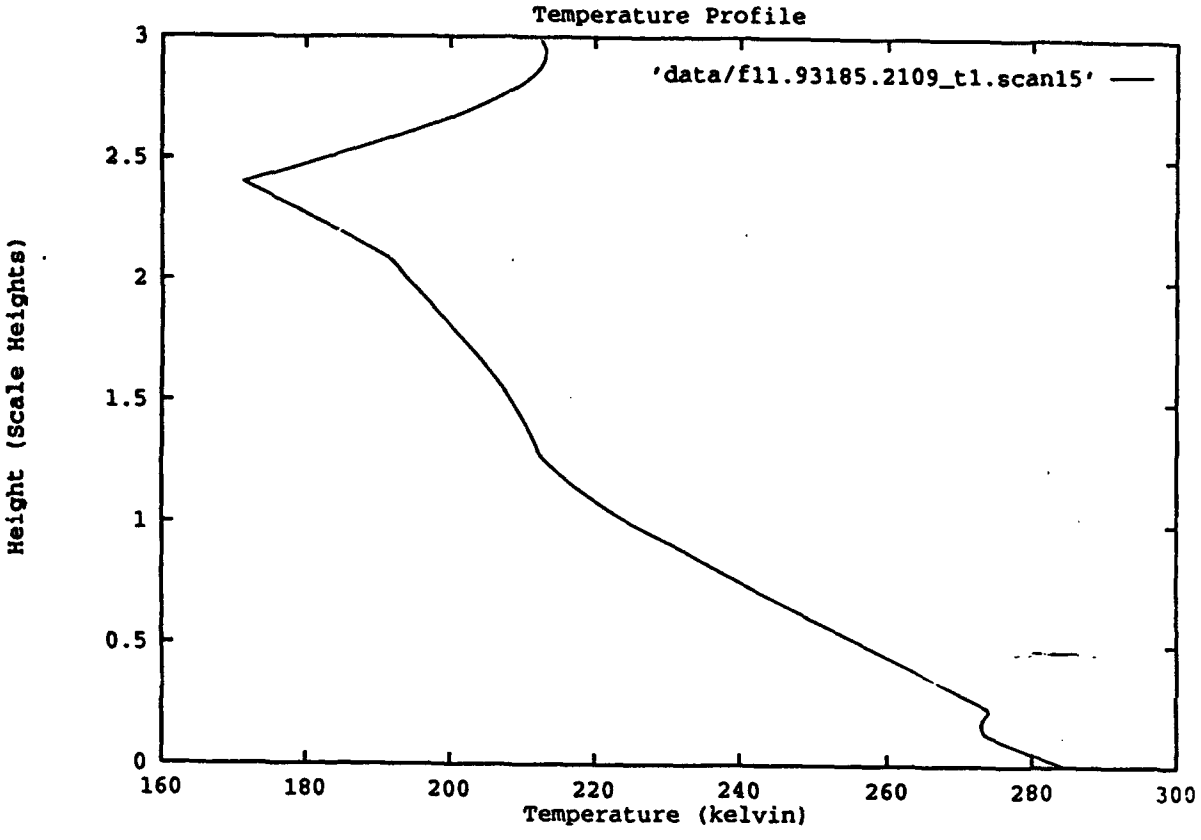

Figure 13: Typical atmospheric temperature profile obtained from the present version of the DI algorithm applied to DMSP SSM/T-1 data. For further information see the caption to Figure 12.

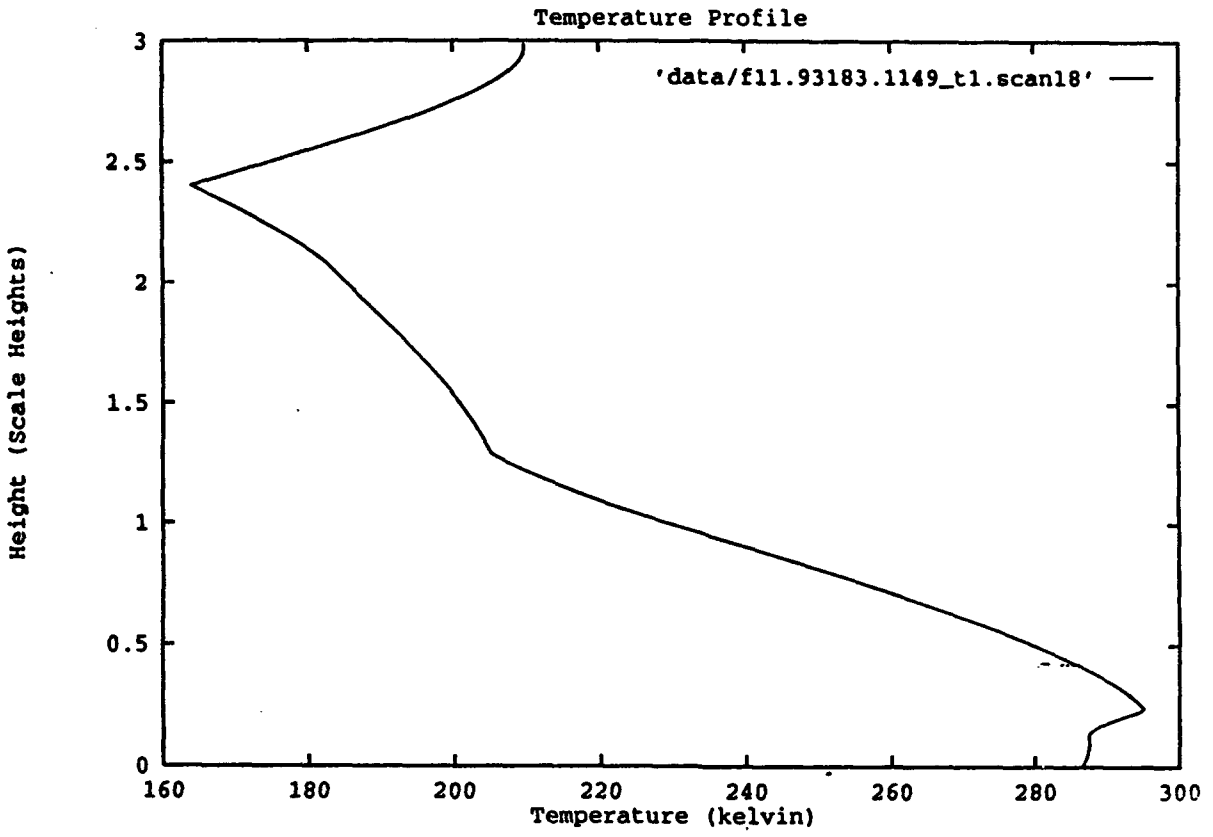

Figure 14: Typical atmospheric temperature profile obtained from the present version of the DI algorithm applied to DMSP SSM/T-1 data. For further information see the caption to Figure 12.

calculate high moments of the atmospheric weight function, which are required for obtaining high order inversion coefficients. Computational tools are available at Phillips Laboratory for computing atmospheric weight functions.

- Weight functions need to be obtained for off-nadir scan angles. Such weight functions can be calculated using the computational tools available at Phillips Laboratory. Inversion coefficients for these off-nadir scan angles can then be computed and temperature profiles obtained. It will then be possible to reduce the entire set of DMSP SSM/T-1 observations, not only those obtained as nadir scans.

- Further research will be conducted on the incorporation of measurements of the ground temperature in obtaining improved temperature profiles. It is expected that the primary source of data for an operational system would be an instrument on the same satellite which would obtain a measurement of the ground surface temperature through a channel which has an atmospheric weight function with the maximum possible area at ground level. We will investigate the suitability of DMSP SSM/I data for this purpose. It should be noted that any practical window channel will be subject to significant meteorological variability, for example due to ground level haze. Research will need to be directed to assessing the importance of these variations in the weight function of such a window channel for the determination of low altitude values of the atmospheric temperature. When possible, we shall also investigate the effects of including ground temperatures obtained from ground observations. (See following item.)

- It is very important to obtain ground truth temperature profiles for comparison with the DI temperature profiles computed for DMSP SSM/T-1 data. Thus far, we have no basis for comparison as to the accuracy of the temperature profiles obtained. We intend to identify times when radiosonde observations are available which correspond to overflights of the DMSP satellite. Such opportunities will provide the best possible circumstances for assessing the performance of the DI algorithm. We expect this to be the major element in the data analysis component of this research in the upcoming year.

- We would also like to compare the temperature profiles obtained by the DI algorithm with the present generation of operational atmospheric temperature profiling algorithm. We will investigate whether this may be carried out in the time frame of the next year's research on this grant.

4 Conclusions

During the first year of this research effort we have carried out the first implementation of the Differential Inversion (DI) algorithm in the microwave band. This work has involved the use of DMSP SSM/T-1 microwave radiances, which were reduced to atmospheric temperature profiles.

These initial temperature profiles show some of the necessary features expected of a meteorologically useful algorithm, but the necessary work to more closely approach that goal seems to be clear and is currently underway in the second year of this research effort. Several important issues relevant to successfully obtaining useful temperature profiles using data from the microwave band have been identified.

1. The shape of the atmospheric weight functions in the microwave band is such that the generalized exponential weight function developed by Dr. King is not as useful as it was for work in the infrared band. As a result, it is necessary to work exclusively with empirical weight functions and to calculate the inversion coefficients from the empirical weight functions directly.

2. Inclusion of the radiation contribution from the ground is much more important in the microwave region than it was in our earlier work in the infrared region. We have developed an extension to DI which allows the inclusion of independent determinations of the ground temperature (either from other instruments or from observations at ground stations). This extension is expected to lead to significantly improved atmospheric temperature profiles low in the atmsophere.

3. The variation of weight function shape with altitude is very important for the determination of an accurate temperature profile. We have developed an approach which accounts for a continuous variation of weight function shape through the atmosphere. Based on this approach, it may be possible to develop a fully self-consistent multispectral version of DI. (The original version of DI derives from the monochromatic equation of radiative transfer.)

Work is also underway to extend our work to the case of nonzero nadir scan angle. The weight functions for nonzero nadir scan angle are very asymmetric, but the formalism of DI depends only on the moments of the weight functions, and thus should be able to accommodate these weight functions in a natural fashion.

We expect that our work will continue to improve the atmospheric temperature profiles obtained with the DI algorithm. The most important issue in the research to be carried out over the upcoming year will be the comparison of the DI temperature profiles with ground truth data to objectively evaluate the performance of the DI algorithm in accurately determining atmospheric temperatures. Towards this end, we plan to obtain atmospheric temperature profiles measured by radiosonde when the DMSP satellite is making an overflight. Such comparisons will provide the best opportunity for evaluating the performance of the DI algorithm and identifying areas for improvement.

A Hyperdistributions and Convolution Algebras

We introduce a general expansion method for functions whose integral properties are the focus of interest. Our expansion displays in configuration space the properties of the classical moment generating expansion for the Fourier transform of the probability distribution. Our expansion allows consideration of "functions" which are more singular than temperate (*i.e.* Fourier-analyzable) distributions, but that can be represented by infinite series of distributions. We call these "functions" hyperdistributions. We show that hyperdistributions form the appropriate framework for carrying out the process of deconvolution, in fact, we give a straightforward algorithm, the "Bochner algebra"[1], to compute explicitly the convolution inverse of any hyperdistribution.

The theory of hyperdistributions is built on ideas related to Dirac's delta function[3], which is technically a "generalized function". The Dirac delta function, $\delta(x)$, has the normalization

$$\int_{-\infty}^{\infty} \delta(x)\,dx = 1 \tag{66}$$

and the sifting property

$$f(x) = \int_{-\infty}^{\infty} \delta(x - x')f(x')\,dx' \tag{67}$$

for any suitably smooth function, $f(x)$. We term equation (67), " Dirac's identity".

The Dirac delta function is even, *i.e.* $\delta(x - x') = \delta(x' - x)$. Expansion for small x gives

$$\delta(x' - x) = \delta(x') - x\delta'(x') + \frac{x^2}{2!}\delta''(x') + \cdots \tag{68}$$

Here $\delta'(x) = d\delta(x)/dx$, $\delta''(x) = d^2\delta(x)/dx^2$, *etc.* This expansion is valid for test functions that are not merly smooth (C^∞) but real analytic(C^ω). Equation (68) allows one to compute a "local" approximation to $f(x)$, since if we substitute this expansion into Dirac's identity, we recover the usual Taylor expansion of $f(x)$ about $x = 0$,

$$f(x) = f(0) + xf'(0) + \frac{x^2}{2!}f''(0) + \cdots \tag{69}$$

This expansion is local in the sense that it requires derivatives of $f(x)$ at a single point, $x = 0$, and in general has a limited radius of convergence. On the other hand, if we expand $\delta(x - x')$ in small x' we have

$$\delta(x - x') = \delta(x) - x'\delta'(x) + \frac{x'^2}{2!}\delta''(x) + \cdots \tag{70}$$

When this series is substituted into Dirac's identity (67), we obtain

$$f(x) = M^{(0)}\delta(x) - M^{(1)}\delta'(x) + \frac{M^{(2)}}{2!}\delta''(x) + \cdots \tag{71}$$

The coefficients $M^{(n)}$ are defined by

$$M^{(n)} = \int_{-\infty}^{\infty} x^n f(x)\, dx, \tag{72}$$

The $M^{(n)}$s are the moments of the function $f(x)$. Therefore, equation (71) is an expansion of $f(x)$ involving global information about $f(x)$, that is, the moments of the function.

This then may be taken as a *motivation for our* definition of a hyperdistribution as a singular function which may be written in the form,

$$f(x) = \sum_{n=0}^{\infty} \alpha_n \nabla^n \delta(x) \tag{73}$$

The coefficients α_n are given by

$$\alpha_n = (-1)^n \frac{M^{(n)}}{n!} \tag{74}$$

Note that equation (73) is equivalent to the familiar moment generating expansion of probability theory. Rhodes[13] has shown that the sum in (73) can be extended to to range from $-\infty$ to ∞ preserving the algebraic properties of hyperdistributions.

If we have two hyperdistributions f_1 and f_2, their linear combination $\lambda f_1 + \mu f_2$ is also a hyperdistribution (where λ and μ are real coefficients). The pth derivative of a hyperdistribution, $d^p f(x)/dx^p = \nabla^p f(x)$, is a hyperdistribution. Also, the convolution of two hyperdistributions $f_1 * f_2$ is a hyperdistribution. These may all be thought of as the "closure properties" of hyperdistributions.

Hyperdistributions allow us to make an effective computation of the convolution inverse. Given a hyperdistribution f, the desired convolution inverse $\mathrm{Inv}[f]$ satisfies

$$f * \mathrm{Inv}[f] = \delta \tag{75}$$

Here δ represents Dirac's delta function, which is the identity of the convolution operation. We shall show by our construction that $\mathrm{Inv}[f]$ is a hyperdistribution. Writing the convolution explicitly,

$$f * \mathrm{Inv}[f] = \int_{-\infty}^{+\infty} dx'\, f(x')\,\mathrm{Inv}[f(x-x')] = \delta(x) \tag{76}$$

We write $f(x)$ and its inverse, $\mathrm{Inv}[f(x)]$, as

$$f(x) = \sum_n \alpha_n \nabla^n \delta(x) \tag{77}$$

$$\mathrm{Inv}[f(x)] = \sum_n \beta_n \nabla^n \delta(x) \tag{78}$$

Substituting the hyperdistribution series for f and $\mathrm{Inv}f$ we obtain into equation (75) we obtain after some algebra the discrete convolution or Bochner algebra [1]

$$\begin{aligned} 1 &= \alpha_0\beta_0 \\ 0 &= \alpha_0\beta_1 + \alpha_1\beta_0 \\ 0 &= \alpha_0\beta_2 + \alpha_1\beta_1 + \alpha_2\beta_0 \\ &\vdots \end{aligned} \tag{79}$$

and so forth. Thus, we can see that, $\beta_0 = 1/\alpha_0$, $\beta_1 = -\alpha_1/{\alpha_0}^2$, $\beta_2 = {\alpha_1}^2/{\alpha_0}^3 - \alpha_2/{\alpha_0}^2$, *etc.* When energy is conserved $\alpha_0 = 1$ and $\beta_0 = 1$. The coefficients are then determined by

$$\beta_i = -\sum_{j=1}^{i} \beta_{i-j}\alpha_j \quad (i > 0) \tag{80}$$

This completes the computation of the terms of the convolution inverse.

Mikusiński[11] has considered the convolution algebra that arises from considering pairs of functions subject to the rules of operation possessed by the rational numbers considered as

pairs of integers. The pair construction, due to Dedekind, insures that the resulting algebra is an algebraic field. The Mikusiński field contains distributions rather than hyperdistributions and is adequate for its intended purpose, namely the rigorous use of the delta function in Heavyside's operational calculus. Consequently, the Mikusiński field is based on functions defined on the half line as needed for the theory of the Laplace transform[5]. Mikusiński does not consider two, three or higher dimensions. More important Mikusiński calculus is suitable for solutions of differential equations and not for integral equations. In fact Mikusiński does not provide a representation for the convolution inverse which is the main purpose of the hyperdistribution calculus. The algebraic field of hyperdistributions has been shown to arise from the pair construction if the members of the pairs are continuous functions which taper at $-\infty$ and ∞, B. A. Rhodes[13].

B Relation of the Multipole Expansion to Differential Inversion

We start with the familiar Poisson equation of potential theory

$$\nabla^2\phi = \rho. \tag{81}$$

In this appendix, we show that, if we substitute ρ for its hyperdistribution expansion, then the resulting potential is Gauss's multipole expansion for ϕ.

Poisson's equation is rewritten wiht the help of the (infinite domain) Green's function as

$$G(x) = \frac{-1}{4\pi r}, \qquad \nabla^2 G(x) = \delta(x) \tag{82}$$

We can then rewrite Poisson's equation as

$$\phi = G * \rho. \tag{83}$$

Introduce Q with the property

$$Q * \rho = \delta \tag{84}$$

Convolving both sides of (??) with Q and using the commutative and associative properties of the $*$ product, we find that

$$\begin{aligned} Q * \phi &= Q * (G * \rho) && (85)\\ &= (Q * \rho) * G && (86)\\ &= G. && (87) \end{aligned}$$

Use the differential inversion solution (Eddington-King formula) to solve this equation for ϕ in terms of the given G

$$\phi(x) = \sum_{k=0}^{\infty} \alpha_k \nabla^k G(x), \tag{88}$$

which is Gauss's multipole series with coefficients

$$\alpha_k = \frac{(-1)^k}{k!} \int_{-\infty}{}^{+\infty} x^{\otimes k} \rho(x)\, d^3x, \tag{89}$$

where $\otimes$ denotes tensor product and where $x^{\otimes k}$ is the tensor power of the 3-vector x. Substituting this equation into (??), we find a familiar expression

$$\phi(x) = -\sum_{k=0}^{\infty} \frac{(-1)^k}{k!} \left(\int_{-\infty}{}^{+\infty} y^{\otimes k} \rho(y)\, d^3y \right) \otimes \nabla^k \frac{+1}{4\pi r}, \tag{90}$$

which is a standard result in potential theory.

C Numerical Differentiation by the Vandermonde Matrix Technique

Application of the Differential Inversion (DI) algorithm depends sensitively on accurate evaluation of derivatives of the known function. In general, accurate evaluation of numerical derivatives is well known to be a difficult problem, particularly for noisy data. We shall develop here a technique which involves the determination of an optimal estimator of the derivative in terms of weighting coefficients applied to samples of the function (or experimental measurements). These coefficients are determined as the solutions of a system of linear equations of Vandermonde type. This method is also known as the "method of undetermined coefficients" and a clear exposition of the method, on which this application is based, has been published by Isaacson and Keller [7].

For appropriate special cases, the coefficients determined by this Vandermonde method reduce to the coefficients of conventional divided difference formulae for estimation of numerical derivatives. However, this method has several important advantages over divided difference formulae: (1) This method provides the best estimator (in the sense of minimizing truncation error) of the numerical derivative at a given order. (2) This estimator applies to the regions between data points as well as to the values at the data points themselves, and so a formally correct continuous representation of the derivatives is obtained. (3) This method is tolerant of unevenly spaced data, unlike divided difference formulae which become unwieldy. (4) The method may readily be used to determine derivatives of high order. (5) Error analysis may be readily carried out in terms of the numerical condition of the Vandermonde matrix.

Begin by considering a general function of one variable, $f(x)$. We want to compute the kth derivative of $f(x)$ at $x = a$, which we denote $f^{(k)}(a)$. The value of $f^{(k)}(a)$ is approximated as a linear combination of samples of the function, $f(x_i)$, $i = 1, \ldots, m$. Say that $f(x)$ has $n+1$ continuous derivatives, where $n+1 \geq m$, and define $h \equiv x_i - a$. Using Taylor's expansion to write

$$f(x_i) = f(a+h_i) \tag{91}$$

$$= f(a) + h_i f^{(1)}(a) + \frac{{h_i}^2}{2!} f^{(2)}(a) + \cdots + \frac{{h_i}^n}{n!} f^{(n)}(a) + \frac{{h_i}^{n+1}}{(n+1)!} f^{(n+1)}(a+\theta_i h_i) \tag{92}$$

where $i = 1, 2, \cdots, m$ and $0 \leq \theta_i \leq 1$ in the usual definition of the remainder term in the Taylor series.

Taking a linear combination of equations (92) with weights, α_i, to be determined:

$$\sum_{i=1}^{m} \alpha_i f(x_i) = \Big(\sum_{i=1}^{m} \alpha_i\Big) f(a) + \Big(\sum_{i=1}^{m} \alpha_i h_i\Big) f^{(1)}(a) + \Big(\sum_{i=1}^{m} \alpha_i {h_i}^2\Big) \frac{f^{(2)}(a)}{2!} + \cdots$$
$$\cdots + \Big(\sum_{i=1}^{m} \alpha_i {h_i}^n\Big) \frac{f^{(n)}(a)}{n!} + \frac{1}{(n+1)!} \sum_{i=1}^{m} \alpha_i {h_i}^{n+1} (a + \theta_i h_i). \quad (93)$$

We are free to choose the coefficients α_i such that the linear combination of the values $f(x_i)$ is the most accurate approximate approximation to $f^{(k)}(a)$, i.e. the best estimator of the derivative. Therefore, we impose the set of m conditions on the m unknown coefficients α_i,

$$\sum_{i=1}^{m} \alpha_i {h_i}^{\nu} = \nu! \delta_{\nu k} \quad (94)$$

where $\nu = 0, 1, \ldots, m-1$, and $\delta_{\nu k}$ is the Kronecker delta function. Generally speaking, we are interested in the cases when $k \geq 1$, since we shall demonstrate below that $f^{(0)}(a)$ merely gives the value of an interpolating polynomial for $f(x)$ evaluated at $x = a$.

It is instructive to write out these equations explicitly for several values of ν. For $\nu = 0$,

$$\sum_{i=1}^{m} \alpha_i {h_i}^0 = 0 \quad (95)$$

which gives us the condition $\alpha_1 + \alpha_2 + \cdots + \alpha_m = 0$. For $\nu = 1$,

$$\alpha_1 h_1 + \alpha_2 h_2 + \cdots + \alpha_m h_m = 1!\, \delta_{1k}. \quad (96)$$

For $\nu = 2$,

$$\alpha_1 {h_1}^2 + \alpha_2 {h_2}^2 + \cdots + \alpha_m {h_m}^2 = 2!\, \delta_{2k}. \quad (97)$$

and so forth until for $\nu = m - 1$,

$$\alpha_1 {h_1}^{m-1} + \alpha_2 {h_2}^{m-1} + \cdots + \alpha_m {h_m}^{m-1} = (m-1)!\, \delta_{m-1,k}. \quad (98)$$

By examining the equations shown above, we note that the system of equations given by equation (93) may be expressed in matrix form as:

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ h_1 & h_2 & \cdots & h_m \\ {h_1}^2 & {h_2}^2 & \cdots & {h_m}^2 \\ \vdots & \vdots & \ddots & \vdots \\ {h_1}^{m-1} & {h_2}^{m-1} & \cdots & {h_m}^{m-1} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ k! \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad (99)$$

It is clear that this system of equations determining the coefficients $\alpha_1, \ldots, \alpha_m$, has a unique solution since the coefficient matrix is of Vandermonde type and hence nonsingular, provided the h_i's are all distinct. In practice, however, high order Vandermonde systems may be numerically ill-conditioned and require specialized algorithms for their numerical solution. Leaving aside for the moment questions of numerical ill-conditionedness, a necessary and sufficient condition for equation (99) to have a solution is for $m > k$, i.e. that the system be nonhomogeneous. This condition corresponds to the well known requirement for determination of derivatives using finite differences that the determination of a kth derivative requires more than k data points.

With the solution of equation (99) substituted in equation (94), we obtain for the estimate ..e kth derivative (recalling that $n + 1 \geq m$),

$$f^{(k)}(a) = \sum_{i=1}^{m} \alpha_i f(x_i) - \frac{1}{m!}\Big(\sum_{i=1}^{m} \alpha_i {h_i}^m\Big) f^{(m)}(a) - \cdots$$
$$\cdots - \frac{1}{n!}\Big(\sum_{i=1}^{m} \alpha_i {h_i}^m\Big) f^{(n)}(a) - \frac{1}{(n+1)!}\Big(\sum_{i=1}^{m} \alpha_i {h_i}^{n+1}\Big) f^{(n+1)}(a + \theta_i h_i) \quad (100)$$

This result is immediate if we note that in order for the α_i's to solve equation (99),

$$\sum_{i=1}^{m} \alpha_i {h_i}^n = 0 \quad (101)$$

for all $n \neq k$.

Several advantages are obtained by this approach:

1. Equation (100) is by construction the most accurate approximation to $f^{(k)}(a)$ using the available data.

2. The data are not required to be evenly spaced.

3. a may be chosen with the set of x_i's as required and our formula automatically evaluates a derivative estimate equivalent to the corresponding centered, forward, or backwards divided difference as required.

4. a need not be a location of one of the measurements, x_i. We can thus obtain a continuous representation of the derivative of a function, not simply a set of derivative estimates evaluated at the x_i's. The curve obtained has the character of a high-order spline curve.

We now specialize our discussion to the requirements of Differential Inversion (DI). We consider a set of image intensities $\phi(z_i)$, with z_i the location of the ith pixel. We shall have a total of m pixels available, and we wish to obtain a kth derivative at a value ζ (corresponding to a in the previous calculation). Note that $k \leq m-1$. Now

$$h_i = z_i - \zeta \tag{102}$$

and we can set up the Vandermonde system of equations for the weighting coefficients α_i, analogous to equation (99).

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ (z_1-\zeta) & (z_2-\zeta) & \cdots & (z_m-\zeta) \\ (z_1-\zeta)^2 & (z_2-\zeta) & \cdots & (z_m-\zeta)^2 \\ \vdots & \vdots & \ddots & \vdots \\ (z_1-\zeta)^{m-1} & (z_2-\zeta)^{m-1} & \cdots & (z_m-\zeta)^{m-1} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ k! \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{103}$$

Note that when ζ lies on one of the z_i values, say z_j, the only consequence for the system is that the jth column has all zero entries, except in the first row. (The first row must remain all ones, as a result of the constraint of continuity on the behavior of the system.) The coefficient matrix then looks like:

$$\begin{pmatrix} 1 & \cdots & 1 & 1 & 1 & \cdots & 1 \\ (z_1-\zeta) & \cdots & (z_{j-1}-\zeta) & 0 & (z_{j+1}-\zeta) & \cdots & (z_m-\zeta) \\ (z_1-\zeta)^2 & \cdots & (z_{j-1}-\zeta)^2 & 0 & (z_{j+1}-\zeta)^2 & \cdots & (z_m-\zeta)^2 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ (z_1-\zeta)^{m-1} & \cdots & (z_{j-1}-\zeta)^{m-1} & 0 & (z_{j+1}-\zeta)^{m-1} & \cdots & (z_m-\zeta)^{m-1} \end{pmatrix} \tag{104}$$

After solving equation (103) for the $\alpha_1, \ldots, \alpha_m$ values, we may then explicitly obtain the estimate of the kth derivative of the image intensity at ζ,

$$\begin{aligned} \phi^{(k)}(\zeta) \;=\; & \sum_{i=1}^{m} \alpha_i \phi(z_i) - \frac{1}{m!}\Big(\sum_{i=1}^{m} \alpha_i {h_i}^m\Big)\phi^{(m)}(\zeta) - \frac{1}{(m+1)!}\Big(\sum_{i=1}^{m} \alpha_i {h_i}^{m+1}\Big)\phi^{(m+1)}(\zeta) - \cdots \\ & \cdots - \frac{1}{n!}\Big(\sum_{i=1}^{m} \alpha_i {h_i}^{n}\Big)\phi^{(n)}(\zeta) - \frac{1}{(n+1)!}\sum_{i=1}^{m} \alpha_i {h_i}^{n+1}\phi^{(n+1)}(\zeta + \theta_i h_i) \end{aligned} \tag{105}$$

All terms except the first on the right hand side of equation (105) represent contributions to the truncation error in the estimation of $\phi^{(k)}(\zeta)$. Note that inclusion of more data points (larger m) raises the order of the truncation error, as one expects on the basis of the Taylor series derivation of estimates of the radiance derivative.

The radiance estimates from equation (105) are then substituted directly into the DI formula for $B(\zeta)$:

$$B(\zeta) = \lambda_0\phi(\zeta) + \lambda_1\phi^{(1)}(\zeta) + \lambda_2\phi^{(2)}(\zeta) + \cdots. \tag{106}$$

If a fixed set of data points is used for estimating the derivatives, the Vandermonde coefficient matrix need be evaluated only once, at some given ζ value. Each successive intensity derivative is then obtained by multiplication of the inverted Vandermonde matrix against a column vector of length m. Since the evaluation of $\phi^{(k)}(\zeta)$ [see eq. (105], given the $\alpha_1, \ldots, \alpha_m$ values, also has the character of an inner product operation, evaluation of the radiance derivatives required in equation (106) (at a given value of ζ) requires inversion of a single Vandermonde matrix and two inner product operations. If a large data set is to be reduced, and each scan will be represented at the same set of ζ values, storing the inverted Vandermonde matrices corresponding to each ζ value will result in substantial improvements in execution time. It should also be noted that this algorithm lends itself to implementation in massively parallel computers.

An important additional advantage of the Vandermonde approach is that it is relatively straightforward to obtain a set of intensity derivative estimates which all have the same order of truncation error. This is important for DI since each term in equation (106) must have a consistent order of truncation error in order that the determination of $B(\zeta)$ have as high an order of truncation error as possible. This is comparatively difficult to arrange in practice using finite difference formulae.

Lastly, as an illustrative example, we shall show how the Vandermonde approach may be used to generate the standard divided difference formulae. Consider points x_1, x_2, x_3 with corresponding function values (measurements) $f(x_1)$, $f(x_2)$, $f(x_3)$. Say that we wish to obtain estimates for $f(x_2)$, $f^{(1)}(x_2)$, and $f^{(2)}(x_2)$. In this case, $m = 3$, and the corresponding Vandermonde matrix is

$$\begin{pmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} \tag{107}$$

The inverse of this matrix is:

$$\begin{pmatrix} 0 & -1/2 & 1/2 \\ 1 & 0 & -1 \\ 0 & 1/2 & 1/2 \end{pmatrix} \tag{108}$$

For the case $k = 0$ we then construct the matrix product,

$$\begin{pmatrix} 0 & -1/2 & 1/2 \\ 1 & 0 & -1 \\ 0 & 1/2 & 1/2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \tag{109}$$

i.e. our estimate for the zeroth derivative is simply $f(x_2)$, as expected. For $k = 1$, we expect a nontrivial result. The α_i's are again found by

$$\begin{pmatrix} 0 & -1/2 & 1/2 \\ 1 & 0 & -1 \\ 0 & 1/2 & 1/2 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} = \begin{pmatrix} -1/2 \\ 0 \\ 1/2 \end{pmatrix} \tag{110}$$

which implies

$$f^{(1)}(x_2) = \frac{f(x_3) - f(x_1)}{2} \tag{111}$$

which is just the centered divided difference estimate for the first derivative. Finally, for $k = 2$,

$$\begin{pmatrix} 0 & -1/2 & 1/2 \\ 1 & 0 & -1 \\ 0 & 1/2 & 1/2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} = \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix} \tag{112}$$

which implies

$$f^{(2)}(x_2) = \frac{f(x_3) - 2f(x_2) + f(x_1)}{1^2} \tag{113}$$

which is the centered divided difference formula for $f^{(2)}(x_2)$.

D Differential Inversion Software Package

D.1 System Design

The following section gives the file structures and the system flowchart for the application of DI to the problem of remote temperature sensing.

The problem here is to design an algorithm at the system level to take radiances and the weighting function as inputs, with some fixed parameters and compute an output temperature profile. There are certain parameters we need to consider when we go about this process.

The weighting function we have here can be either the empirical weighting function that is obtained from the SSM/T microwave sounder or for example, the analytic weighting function as given by King [8]. We have decided to take advantage of the intrinsic flexibility of DI, owing to its formulation in terms of the moments of weight functions, to develop a general software package which can be applied to a variety of instruments. The software must accept specifications of the instrument in an appropriate form. The number of channels the system with the peaking height for these channels must also be specified. Since we need to normalize these peaking heights it is also necessary to specify the scale height of the atmosphere. All the above mentioned parameters are provided to the system using a parameter file which has all these values stored in it. The structure of the parameter file is as shown below:

```
Parameters File :

WTFN < 0 or 1 >          Specifies that weighting function is
                         "0"-Empirical \& "1"-Jean King
INST <string>            Type of instrument
SCLHT <value>            The scale height
CHANNELS <number>        The no. of channels in the system.
ZB <value> HZ <value>    Z-bar and freq. value of channel 1.
ZB <value> HZ <value>    Z-bar and freq. value of channel 2.
   ...
   ...
```

```
ZB <value> HZ <value>  Z-bar and freq. value of channel n.
```

The program would read this parameter file and get the necessary values for the computations. The string 'HZ' specifies the frequency is in Hertz. The other possible entries here could be 'GHz' or 'INV_CM'.

The datafile would consist of the radiance values for the different channels for all the scans that are done by the instrument. The format of the data file is as provided by the respective instrument. We receive files from Atmospheric and Environmental Research, Inc., which is operating at Phillips Laboratory under contract to the United States Air Force. Atmospheric and Envirnomental Research, Inc. unbundles files of measured system temperatures (which are stored in a proprietary file format) and provides them to us in a computer which we can transfer to Boston University over Internet.

D.2 Program Operation

The command that would be typed at the system prompt (unix prompt) by the user is

```
% di -P<parameter file> -D<data file> -S<# of scans>
```

The suffixes "-P,-D,-S" allows the system to interpret the information that is entered at the prompt. If all the three inputs are not given to the system then the computation process will be aborted.

The various steps that are executed in order as a part of the system run are summarised below.

[1] Check if all the three parameters are entered at the command prompt.

[2] Read in parameter file and do necessary validations and also verify if all the necessary inputs are provided. Do cross-validations to see to that the no. of channels entered tally with the no. of $\overline{z}$ values given.

[3] Get the no. of scans from the prompt line.

[4] Determine the weighting function is empirical or the Jean King function and accordingly branch for the computations.

[5] Generate moments and lambdas for all channels. Store the values in a two dimensional array of the form λ[channel][no.]. Thus λ[1][0] will correspond to the value of λ_0 for channel 1.

[6] Read in the values radiances for the one scan.

[7] Use the lambda values and the radiance values and compute B[z].

[8] Invert the function of B[z] to compute T[z].

[9] Repeat steps 6 thru 8 for all the scans.

On executing the above steps we will obtain a temperature profile for each scan that is done by the instrument.

E Listing of Differential Inversion Software

The following pages are a listing of the C code currently used in the implementation of the Differential Inversion algorithm as applied to DMSP SSM/T-1 data.

```
/*-------------------------------------------------------------------------*/
/*  Program Name : DI.C                                                    */
/*-------------------------------------------------------------------------*/
/*  Purpose : To provide a user interface for using the DI algorithm as    */
/*            as applied to remote temperature sensing.                    */
/*  Files read : Parameter file, Data file, and Weight Function files      */
/*-------------------------------------------------------------------------*/
/*  Functions called by DI.c :                                             */
/*-------------------------------------------------------------------------*/
/*    Function Name                                  Purpose               */
/*-------------------------------------------------------------------------*/
/*   param()                        Process the parameter file             */
/*   read_chan_info()               Process parameter file and get channel */
/*                                  information pertaining to freq, and    */
/*                                  peaking height.                        */
/*   fit_freq()                     Fits a curve to freq vs. Zbar(peaking ht) */
/*-------------------------------------------------------------------------*/
/* If the weight function is the Eddington-King function :                 */
/*-------------------------------------------------------------------------*/
/*   calc_mom2()                    Calculates the moments for the weight  */
/*                                  function.                              */
/*   gen_lambdas2()                 Calculates the lambdas using the moments */
/*                                  obtained from calc_mom2().             */
/*-------------------------------------------------------------------------*/
/* If the weight functions used are the DMSP SSM/T-1 weight functions.     */
/*-------------------------------------------------------------------------*/
/*   gb_calc_mom()                  Calculates the moments using the SSM/T */
/*                                  weight functions, considering the ground */
/*                                  boundary conditions.                   */
/*   gen_lambdas()                  Calculates the lambdas using the moments */
/*                                  obtained from gb_calc_mom().           */
/*-------------------------------------------------------------------------*/
/*   calc_deriv()                   Fits a curve to the radiance intensity */
/*                                  and Zbar values and also computes the  */
/*                                  derivatives of R(z) at different heights. */
/*   calc_B() / calc_B2()           Computes the planck values for different */
/*                                  heights using the derivative values and */
/*                                  computed lambda values                 */
/*   B_T()                          Converts the planck values to temperature */
/*                                  values.                                */
/*-------------------------------------------------------------------------*/

/*-- Include Files --*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*--- Define Statements --*/

#define C 3e10
#define MAX_ZETA 200
#define COEFF 6
#define SCAN_TO_VIEW 15
#define DER_TO_USE 4

/*----- Variable Definitions ---*/

char temp[20],temp1[20],temp2[10],temp3[10];
int i,j,k,len,noscn,win,derv;
```

```
char pfnm[15],dfnm[25],scan[5];
char filnm[20];
FILE *fptr,*fp1;
char FLAG;

int wtflag,chan,m_nos;
double freq_mult_fact,sclht,mom,zbar[100],freq[100],rad[100],sigma[100];
double deriv_data[MAX_ZETA][100],zeta_data[MAX_ZETA],freq_data[MAX_ZETA];
double planck[MAX_ZETA],temper[MAX_ZETA];
double moments[100],em_mom[100][100];
double lamb[100],em_lamb[100][100],gb_lamb[MAX_ZETA][100];
char inst[10],outfile[40];

/*-------------------------------------------------------------------------*/
/*     Routine to read the channel information from the Parameter file     */
/*-------------------------------------------------------------------------*/

read_chan_info(cn)
int cn;
{
for(i=0;i<cn;i++)
 { fscanf(fptr,"%s %lf %s %lf\n",temp,&zbar[i],temp2,&freq[i]);
     /*--- Reading the frequency and zbar values --*/
  if (strcmp(temp,"ZB") != 0)
   { printf("Peaking ht. info. missing .. \n");
     exit(0);
   };

  /*-- Converting the frequency value to units of  Inverse Cm ---*/

  if (strcmp(temp2,"Hz") != 0)
   { if (strcmp(temp2,"GHz") != 0)
      { if (strcmp(temp2,"InvCm") != 0)
         { printf("Invalid frequency info ..\n");
           exit(0);
         }
       else freq_mult_fact = 1.0;
      }
     else freq_mult_fact = 1.0e+9/C;
   }
  else freq_mult_fact = 1/C;


  freq[i] *= freq_mult_fact;

  printf("  %4d %12.2lf %15.6lf\n",i+1,zbar[i],freq[i]);
 };
printf("\n");
}

/*-------------------------------------------------------------------------*/
/*--                      Routine to process parameter file              --*/
/*-------------------------------------------------------------------------*/

param(filnm)
char filnm[15];
{
fptr = fopen(filnm,"r");

if (fptr == NULL)
```

```
{ printf("Invalid parameter filename .. \n\n");
  exit(0);
};

/*--- Verifying the Weight function type --*/

fscanf(fptr,"%s %d",temp,&wtflag);
if (strcmp(temp,"WTFN") != 0)
{ printf("Wrong entry in parameter file .. \n\n");
  exit(0);
};

if (wtflag == 0)
   { printf("                    ---------------------------------\n");
     printf("                   |  Empirical Weighting function  |\n");
   }
else if (wtflag == 1)
       { printf("\n                      ---------------------------------\n");
         printf("                     |  Jean King Weighting function  |\n");
        }
      else {printf("Error : Cannot recognise weighting function type..\n\n");
           exit(0);
          };
printf("                    ---------------------------------\n\n");

/*-- Getting other information from the Parameter file --*/

while(feof(fptr) == 0)
 { fscanf(fptr,"%s %s",temp,temp1);
   if (feof(fptr) != 0) break;

   if (strcmp(temp,"ZB") == 0)
   { printf("No. of channels missing .. \n");
     exit(0);
   };

   if (strcmp(temp,"INST") == 0)
   { strcpy(inst,temp1);
     if (strcmp(inst,"NIL")==0)
       { printf("Invalid Instrument name ..\n");
         exit(0);
       };
     printf("Instrument : %s\n",inst);
   };

   if (strcmp(temp,"SCLHT") == 0)
   { sclht = atof(temp1);
     printf("Scale Height : %lf\n",sclht);
   };

   if (strcmp(temp,"CHANNELS") == 0)
   { chan = atoi(temp1);
     if (chan==0)
       { printf("Invalid no. of channels ..\n");
         exit(0);
       };
     printf("Number of Channels : %d \n\n",chan);
     printf("  Channel     Z_Bar     Wave Number(1/cm)\n");
     printf("  ------------------------------------\n");
     read_chan_info(chan);
```

```
   };

 };
}

/*-------------------------------------------------------------------------*/
/*                              Main Program                               */
/*-------------------------------------------------------------------------*/

main(argc,argv)
int argc;
char **argv;
{

FILE *fp,*tptr;
double zb,tempo;
int chan1,line,cal1,cal2;

/*-- Checking if program has 4 arguments --*/

if (argc != 4)
{
  printf("Syntax Error .. \n");
  printf("DI -P<parameter file> -D<data file> -S<no. of scans>\n\n");
  exit(0);
};

strcpy(scan,"NIL");
strcpy(dfnm,"NIL");
strcpy(pfnm,"NIL");
strcpy(inst,"NIL");

/*-- Reading in the arguments w.r.t the flags P,D and S --*/

while(*argv != NULL)
  { strcpy(temp,*argv++);
    len = strlen(temp);
    if (temp[1] == 'P')
     { for (i=0;i<=len-2;i++) pfnm[i] = temp[i+2];
       pfnm[i] = '\0';
     };
    if (temp[1] == 'D')
     { for (i=0;i<=len-2;i++) dfnm[i] = temp[i+2];
       dfnm[i] = '\0';
     };
    if (temp[1] == 'S')
     { for (i=0;i<=len-2;i++) scan[i] = temp[i+2];
       scan[i] = '\0';
     };
  };

noscn = atoi(scan);

/*-- Checking if all arg. reqd. were given --*/

if ( (strcmp(scan,"NIL") == 0) | (strcmp(dfnm,"NIL") == 0)
         | (strcmp(pfnm,"NIL") == 0) | (noscn == 0) )
{
  printf("Syntax Error .. \n");
  printf("DI -P<parameter file> -D<data file> -S<no. of scans>\n\n");
```

```
  exit(0);
};

printf ("\n Parameter file name : %s\n Data file name : %s\n No. of Scans : %d\n\n",pfnm,df


/*-- Processing parameter file --*/

param(pfnm);

/*--- Convert to scale heights ---*/

for (i=0;i<chan;++i)
  zbar[i] /= sclht;

printf("Press <ENTER> to continue ...");
getchar();

system("clear");

/*-----------------------------------------------------------------------*/

printf("\nFIT A CURVE TO THE FREQUENCY VALUES  ");
fit_freq(chan,COEFF,zbar,freq,freq_data,zeta_data);
printf("CURVE FITTING SUCCESSFUL ..\n");

/*-----------------------------------------------------------------------*/

printf("\n CALCULATING MOMENTS AND LAMBDA ..    \n");

if (wtflag == 1)
 { printf("\nEnter value of m and no. of moments : ");
   scanf("%lf  %d",&mom,&m_nos);
   printf("\nPlease Wait ..... ");
   calc_mom2(mom,moments);
   gen_lambdas2(m_nos,moments,lamb);
  }
else
 { m_nos = 10;
   printf("Reading the weight files of the instrument %s ...\n",inst);
   gb_calc_mom(chan,zbar,sclht,em_mom);
   gen_lambdas(chan,m_nos,em_mom,em_lamb);
 }

printf("\nLAMBDA CALCULATION COMPLETED ... \n");

/*-----------------------------------------------------------------------*/

fp = fopen(dfnm,"r");
if (fp == NULL)
{ printf("Invalid data filename .. \n\n");
  exit(0);
};


/*-----------------------------------------------------------------------*/
/*       Get radiances for all the channels in multiple scan mode        */
/*-----------------------------------------------------------------------*/
```

```
for (line=0;line<noscn;++line)
{FLAG=0;
 printf("\nScan no. :  %d \n",line+1);
 if (strcmp(pfnm,"par.dat") == 0) chan1=chan; else chan1=chan+1;

 for(i=1;i<chan;++i)
  { fscanf(fp,"%lf ",&rad[i]);
    if (rad[i] == 0.0) FLAG=1;
    printf("%10.6lf",rad[i]);
  };
 printf("\n");
 if (FLAG==1) continue;


 rad[0] = 0.006396;  /* Using the this value of radiance for the */
                     /* ground channel - Computed for Brightness */
                     /* temperature of 285 kelvin                */

 tempo = rad[5];     /* Shifting the radiance values w.r.t the   */
 rad[5] = rad[6];    /* zbar values in the parameter file.       */
 rad[6] = rad[7];
 rad[7] = tempo;


/*-------------------------------------------------------------------------*/


chan1 = chan;

fp1 = fopen("sigma.dat","r");
for (i=0;i<chan;i++)
  fscanf(fp1,"%lf",&sigma[i]);
fclose(fp1);

calc_deriv(chan,COEFF,zbar,rad,sigma,deriv_data,zeta_data);

/*-------------------------------------------------------------------------*/

if (wtflag==1)
  calc_B2(DER_TO_USE,lamb,deriv_data,planck,zeta_data);
else
  calc_B(chan,COEFF,em_lamb,deriv_data,planck,zeta_data,zbar);

/*-------------------------------------------------------------------------*/


B_T(chan,freq_data,zbar,zeta_data,planck,temper);

/*-------------------------------------------------------------------------*/


strcpy(outfile,dfnm);
outfile[strlen(outfile)-4] = '\0';
strcat(outfile,".scan");

if (line < 9)
  { outfile[strlen(outfile)] = line+49;
    outfile[strlen(outfile) + 1] = '\0';
  }
else
```

```
  { k = strlen(outfile);
    cal1=cal2=line+1;
    cal1 = (int)(cal1/10);
    cal2 -= cal1*10;
    outfile[k] = cal1+48;
    outfile[k + 1] = cal2+48;
    outfile[k + 2] = '\0';
  };

tptr = fopen(outfile,"w");
for (i=0;i<MAX_ZETA;++i)
  fprintf(tptr,"%lf  %lf\n",temper[i],zeta_data[i]);

fclose(tptr);

for(i=0;i<70;++i) printf("-");
printf("\n");

} /* For loop - scans */
}

/*-------------------------------------------------------------------------*/
/*                          End of Program                                 */
/*-------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : LFIT.c                                                  */
/*-------------------------------------------------------------------------*/
/*  Purpose : This program has the functions lfit(),calc_deriv() and       */
/*            fit_freq(). The lfit() function calls gaussj() and covsrt(). */
/*            The functions lfit(),gaussj() and covsrt() are used as given */
/*            in the book Numerical Recipes in C.                          */
/*-------------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>

#define MAX_ZETA 200

static double sqrarg;
#define SQR(a) (sqrarg=(a),sqrarg*sqrarg)

double pow();

void lfit(x,y,sig,ndata,a,ma,lista,mfit,covar,chisq,funcs)
int ndata,ma,lista[10],mfit;
double x[100],y[100],sig[100],a[10],**covar,*chisq;
void (*funcs)();        /* ANSI: void (*funcs)(double,double *,int); */
{
        int k,kk,j,ihit,i;
        double ym,wt,sum,sig2i,**beta,*afunc;
        void gaussj(),covsrt(),nrerror(),free_matrix(),free_vector();
        double **matrix(),*vector();

        beta=matrix(1,ma,1,1);
        covar=matrix(1,ma,1,ma);
        afunc=vector(1,ma);
        kk=mfit+1;


        for (j=1;j<=ma;j++) {
                ihit=0;
                for (k=1;k<=mfit;k++)
                        if (lista[k] == j) ihit++;

                if (ihit == 0)
                        lista[kk++]=j;
                else if (ihit > 1) nrerror("Bad LISTA permutation in LFIT-1");
        }

        if (kk != (ma+1)) nrerror("Bad LISTA permutation in LFIT-2");
        for (j=1;j<=mfit;j++) {
                for (k=1;k<=mfit;k++) covar[j][k]=0.0;
                beta[j][1]=0.0;
        }
        for (i=1;i<=ndata;i++) {
                (*funcs)(x[i],afunc,ma);
                ym=y[i];
                if (mfit < ma)
                        for (j=(mfit+1);j<=ma;j++)
                                ym -= a[lista[j]]*afunc[lista[j]];
                sig2i=1.0/SQR(sig[i]);
                for (j=1;j<=mfit;j++) {
                        wt=afunc[lista[j]]*sig2i;
                        for (k=1;k<=j;k++)
```

```
                                covar[j][k] += wt*afunc[lista[k]];
                        beta[j][1] += ym*wt;
                }
        }
        if (mfit > 1)
                for (j=2;j<=mfit;j++)
                        for (k=1;k<=j-1;k++)
                                covar[k][j]=covar[j][k];
        gaussj(covar,mfit,beta,1);
        for (j=1;j<=mfit;j++) a[lista[j]]=beta[j][1];

        *chisq=0.0;
        for (i=1;i<=ndata;i++) {
                (*funcs)(x[i],afunc,ma);
                for (sum=0.0,j=1;j<=ma;j++) sum += a[j]*afunc[j];
                *chisq += SQR((y[i]-sum)/sig[i]);
        }
        covsrt(covar,ma,lista,mfit);
        free_vector(afunc,1,ma);
        free_matrix(beta,1,ma,1,1);
}

#undef SQR

/*-------------------------------------------------------------------------*/

funct(zb2,afunc2,len)
double zb2,afunc2[10];
int len;
{

int i;
double pow();

for (i=1;i<=len;i++)
   { if (i >= 2) afunc2[i] = pow(zb2,(double)(i-1));
       else afunc2[i] = 1.0;
   }
}

/*-------------------------------------------------------------------------*/
/* calc_deriv() - Called by DI.c to compute derivatives                    */
/*-------------------------------------------------------------------------*/

calc_deriv(chan,COEFF,zb,rad,sigma,deriv,zeta)
int chan,COEFF;
double zb[100],rad[100],deriv[MAX_ZETA][100],zeta[MAX_ZETA],sigma[100];
{
int i,j,k,maxi,list[10];
double a[10],chisqr[10],cvr[10][10];
double max,incr;
FILE *fptr,*fp;

for (i=1;i<=COEFF;i++) list[i] = i;

for (i=chan-1;i>=0;--i)
  { zb[i+1] = zb[i];
    rad[i+1] = rad[i];
    sigma[i+1] = sigma[i];
  }
```

```
lfit(zb,rad,sigma,chan,a,COEFF,list,COEFF,cvr,chisqr,funct);

/**-- Compute the zeta values by dividing into 200 points --*/

  max = zb[1];
  maxi=1;
  for (i=1;i<=chan;++i)
    if (zb[i] > max)
    { max = zb[i];
       maxi = i;
     };
  incr = (zb[maxi] - zb[0])/(double)MAX_ZETA;

  for (k=0;k<MAX_ZETA;++k)
    zeta[k] = (incr * (double)k) + zb[1];


/**-- Computing Derivatives --**/
/**-- Derivative matrix starts from [0,0] position --**/

a[COEFF+1] = 0.0;

 for(i=0;i<COEFF;i++)
 {
  for (k=1;k<=MAX_ZETA;k++)
   { deriv[k-1][i] = a[1];
     for (j=1;j<=COEFF;j++)
       deriv[k-1][i] += a[j+1] * pow(zeta[k-1],(double)j);
   }

  /**--------- Shift a[] array ---------**/

  for(j=1;j<=COEFF;j++)
     a[j]=(double)j * a[j+1];
 }

for (i=0;i<chan;i++)
  { zb[i] = zb[i+1];
    rad[i] = rad[i+1];
    sigma[i] = sigma[i+1];
  }
}

/*-------------------------------------------------------------------------*/
/* fit_freq() : Called by DI.c to fit a curve to frequency and Zbar.       */
/*-------------------------------------------------------------------------*/

fit_freq(chan,COEFF,zb,freq,ffreq,zeta)
int chan,COEFF;
double zb[100],freq[100];
double ffreq[MAX_ZETA],zeta[MAX_ZETA];
{
int i,j,k,maxi,list[10];
double a[10],chisqr[10],cvr[10][10],sigma[100];
double max,incr;
FILE *fptr;

for (i=1;i<=COEFF;i++)
  { list[i] = i;
```

```
  }

for (i=chan-1;i>=0;--i)
  { zb[i+1] = zb[i];
    freq[i+1] = freq[i];
    sigma[i+1] = 0.003333333;
  }


lfit(zb,freq,sigma,chan,a,COEFF,list,COEFF,cvr,chisqr,funct);

for (i=1;i<=COEFF;i++) printf("a[%d] = %e\n",i,a[i]);
printf("\n\n");
getchar();


/**-- Compute the zeta values by dividing into 200 points --*/

  max = zb[1];
  maxi=1;
  for (i=1;i<=chan;++i)
    if (zb[i] > max)
    { max = zb[i];
        maxi = i;
     };
  incr = (zb[maxi] - zb[0])/(double)MAX_ZETA;

  for (k=0;k<MAX_ZETA;++k)
    zeta[k] = (incr * (double)k) + zb[1];


for (k=1;k<=MAX_ZETA;k++)
  { ffreq[k-1] = a[1];
     for (j=1;j<=COEFF;j++)
       ffreq[k-1] += a[j+1] * pow(zeta[k-1],(double)j);
  }

fptr = fopen("freqfit","w");
for (i=0;i<MAX_ZETA;i++)
   fprintf(fptr,"%lf   %lf\n",zeta[i],ffreq[i]);
fprintf(fptr,"\n\n");
for (i=1;i<=chan;i++)
   fprintf(fptr,"%lf   %lf\n",zb[i],freq[i]);

fclose(fptr);

for (i=0;i<chan;i++)
  { zb[i] = zb[i+1];
    freq[i] = freq[i+1];
  }
}

/*-------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : GAUSSJ.c                                                */
/*  Purpose : Gauss-Jordan elimination technique - Called by LFIT.c        */
/*-------------------------------------------------------------------------*/

#include <math.h>

#define SWAP(a,b) {double temp=(a);(a)=(b);(b)=temp;}

void gaussj(a,n,b,m)
double **a,**b;
int n,m;
{
        int *indxc,*indxr,*ipiv;
        int i,icol,irow,j,k,l,ll,*ivector();
        double big,dum,pivinv;
        void nrerror(),free_ivector();

        indxc=ivector(1,n);
        indxr=ivector(1,n);
        ipiv=ivector(1,n);
        for (j=1;j<=n;j++) ipiv[j]=0;
        for (i=1;i<=n;i++) {
                big=0.0;
                for (j=1;j<=n;j++)
                        if (ipiv[j] != 1)
                                for (k=1;k<=n;k++) {
                                        if (ipiv[k] == 0) {
                                                if (fabs(a[j][k]) >= big) {
                                                        big=fabs(a[j][k]);
                                                        irow=j;
                                                        icol=k;
                                                }
                                        } else if (ipiv[k] > 1) nrerror("GAUSSJ: Singular M
                                }
                ++(ipiv[icol]);
                if (irow != icol) {
                        for (l=1;l<=n;l++) SWAP(a[irow][l],a[icol][l])
                        for (l=1;l<=m;l++) SWAP(b[irow][l],b[icol][l])
                }
                indxr[i]=irow;
                indxc[i]=icol;
                if (a[icol][icol] == 0.0) nrerror("GAUSSJ: Singular Matrix-2");
                pivinv=1.0/a[icol][icol];
                a[icol][icol]=1.0;
                for (l=1;l<=n;l++) a[icol][l] *= pivinv;
                for (l=1;l<=m;l++) b[icol][l] *= pivinv;
                for (ll=1;ll<=n;ll++)
                        if (ll != icol) {
                                dum=a[ll][icol];
                                a[ll][icol]=0.0;
                                for (l=1;l<=n;l++) a[ll][l] -= a[icol][l]*dum;
                                for (l=1;l<=m;l++) b[ll][l] -= b[icol][l]*dum;
                        }
        }
        for (l=n;l>=1;l--) {
                if (indxr[l] != indxc[l])
                        for (k=1;k<=n;k++)
                                SWAP(a[k][indxr[l]],a[k][indxc[l]]);
        }
```

```
        free_ivector(ipiv,1,n);
        free_ivector(indxr,1,n);
        free_ivector(indxc,1,n);
}

#undef SWAP
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : COVSRT.c                                                */
/*  Purpose : To do computation with the covariance matrix                 */
/*-------------------------------------------------------------------------*/

void covsrt(covar,ma,lista,mfit)
double **covar;
int ma,lista[],mfit;
{
        int i,j;
        double swap;

        for (j=1;j<ma;j++)
                for (i=j+1;i<=ma;i++) covar[i][j]=0.0;
        for (i=1;i<mfit;i++)
                for (j=i+1;j<=mfit;j++) {
                        if (lista[j] > lista[i])
                                covar[lista[j]][lista[i]]=covar[i][j];
                        else
                                covar[lista[i]][lista[j]]=covar[i][j];
                }
        swap=covar[1][1];
        for (j=1;j<=ma;j++) {
                covar[1][j]=covar[j][j];
                covar[j][j]=0.0;
        }
        covar[lista[1]][lista[1]]=swap;
        for (j=2;j<=mfit;j++) covar[lista[j]][lista[j]]=covar[1][j];
        for (j=2;j<=ma;j++)
                for (i=1;i<=j-1;i++) covar[i][j]=covar[j][i];
}
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : FACTORIAL.c                                             */
/*-------------------------------------------------------------------------*/

#include <stdio.h>

int factorial(n)
int n;
{
  int i, product = 1;

  for (i=2;i<=n;++i)
    product *= i;
  return product;
}
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : POLINT.c                                                */
/*  Purpose : Called by the Romberg integrator program                     */
/*-------------------------------------------------------------------------*/

#include <math.h>

void polint(xa,ya,n,x,y,dy)
double xa[],ya[],x,*y,*dy;
int n;
{
        int i,m,ns=1;
        double den,dif,dift,ho,hp,w;
        double *c,*d,*dvector();
        void nrerror(),free_dvector();

        dif=fabs(x-xa[1]);
        c=dvector(1,n);
        d=dvector(1,n);
        for (i=1;i<=n;i++) {
                if ( (dift=fabs(x-xa[i])) < dif) {
                        ns=i;
                        dif=dift;
                }
                c[i]=ya[i];
                d[i]=ya[i];
        }
        *y=ya[ns--];
        for (m=1;m<n;m++) {
                for (i=1;i<=n-m;i++) {
                        ho=xa[i]-x;
                        hp=xa[i+m]-x;
                        w=c[i+1]-d[i];
                        if ( (den=ho-hp) == 0.0)
                          nrerror("Error in routine POLINT");
                        den=w/den;
                        d[i]=hp*den;
                        c[i]=ho*den;
                }
                *y += (*dy=(2*ns < (n-m) ? c[ns+1] : d[ns--]));
        }
        free_dvector(d,1,n);
        free_dvector(c,1,n);
}
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : CALC_MOM2.c                                             */
/*-------------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>

/* This appears reliable up to about a[8], or lambda[8] */

int n;
double m;
FILE *fp;

calc_mom2(mval,mom_data)
double mval,mom_data[10];
{
  int nfact;
  int factorial();
  double a,b,c,d,nterm;
  double mom,moment1,moment2,moment3,moment;
  double W2();
  double func2();
  double trapzd2();
  double midinf2();
  double midexp2();
  double qromb2();
  double qromo2();
  void polint(),nrerror();

  m = mval;

  a = -1.0e30;
  b = -2.0;
  c = 5.0;
  d = 1.0e30;
  for (n=0;n<=9;++n) {
    nfact = factorial(n);
    nterm = 1.0/((double) nfact);
    moment1 = qromo2(func2,a,b,midinf2);
    moment2 = qromb2(func2,b,c);
    moment3 = qromo2(func2,c,d,midexp2);
    mom = moment1 + moment2 + moment3;
    moment = nterm * mom;
    mom_data[n] = moment;
   }
}
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : QROMB2.c                                                */
/*-------------------------------------------------------------------------*/

#include <math.h>

#define EPS 1.0e-6
#define JMAX 20
#define JMAXP JMAX+1
#define K 5

int n;
double m;

double qromb2(func2,a,b)
double a,b;
double (*func2)();
{
        double ss,dss,trapzd2();
        double s[JMAXP+1],h[JMAXP+1];
        int j;
        void polint(),nrerror();

        h[1]=1.0;
        for (j=1;j<=JMAX;j++) {
                s[j]=trapzd2(func2,a,b,j);
                if (j >= K) {
                        polint(&h[j-K],&s[j-K],K,0.0,&ss,&dss);
                        if (fabs(dss) < EPS*fabs(ss)) return ss;
                }
                s[j+1]=s[j];
                h[j+1]=0.25*h[j];
        }
        nrerror("Too many steps in routine QROMB");
}

#undef EPS
#undef JMAX
#undef JMAXP
#undef K
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : TRAPZD2.c                                               */
/*-------------------------------------------------------------------------*/

#define FUNC2(x) ((*func2)(x))

int n;
double m;

double trapzd2(func2,a,b,N)
int N;
double a,b;
double (*func2)();      /* ANSI: float (*func)(float); */
{
        double fna,fnb,fnx;
        double x,tnm,sum,del;
        static double s;
        static int it;
        int j;

        if (N == 1) {
                it=1;
                fnb = FUNC2(b);
                fna = FUNC2(a);
                return (s=0.5*(b-a)*(fna+fnb));
        } else {
                tnm=it;
                del=(b-a)/tnm;
                x=a+0.5*del;
                for (sum=0.0,j=1;j<=it;j++,x+=del) {
                  fnx = FUNC2(x);
                  sum += fnx;
                }
                it *= 2;
                s=0.5*(s+(b-a)*sum/tnm);
                return s;
        }
}

#undef FUNC2(x) ((*func2)(x))
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : QROMO2.c                                                */
/*-------------------------------------------------------------------------*/

#include <math.h>

#define EPS 1.0e-6
#define JMAX 14
#define JMAXP JMAX+1
#define K 5

int n;
double m;

double qromo2(func2,a,b,choose)
double a,b;
double (*func2)();
double (*choose)();     /* ANSI: float choose(float(*)(float),float,float,int); */
{
        int j;
        double ss,dss,h[JMAXP+1],s[JMAXP+1];
        void polint(),nrerror();

        h[1]=1.0;
        for (j=1;j<=JMAX;j++) {
                s[j]=(*choose)(func2,a,b,j);
                if (j >= K) {
                        polint(&h[j-K],&s[j-K],K,0.0,&ss,&dss);
                        if (fabs(dss) < EPS*fabs(ss)) {
                          return ss;
                        }
                }
                s[j+1]=s[j];
                h[j+1]=h[j]/9.0;
        }
        nrerror("Too many steps in rc      QROMO");
}

#undef EPS
#undef JMAX
#undef JMAXP
#undef K
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : MIDEXP2.c                                               */
/*-------------------------------------------------------------------------*/

#include <math.h>

#define FUNC2(x) ((*func2)(-log(x))/(x))

int n;
double m;

double midexp2(func2,aa,bb,N)
int N;
double aa,bb;
double (*func2)();      /* ANSI: float (*funk)(float); */
{
        double x,tnm,sum,del,ddel,b,a;
        static double s;
        static int it;
        int j;

        b=exp(-aa);
        a=0.0;
        if (N == 1) {
                it=1;
                s=(b-a)*FUNC2(0.5*(a+b));
                return s;
        } else {
                tnm=it;
                del=(b-a)/(3.0*tnm);
                ddel=del+del;
                x=a+0.5*del;
                sum=0.0;
                for (j=1;j<=it;j++) {
                        sum += FUNC2(x);
                        x += ddel;
                        sum += FUNC2(x);
                        x += del;
                }
                it *= 3;
                s=(s+(b-a)*sum/tnm)/3.0;
                return s;
        }
}

#undef FUNC2
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : MIDINF2.c                                               */
/*-------------------------------------------------------------------------*/

#define FUNC2(x) ((*funk2)(1.0/(x))/((x)*(x)))

int n;
double m;

double midinf2(funk2,aa,bb,N)
int N;
double aa,bb;
double (*funk2)();      /* ANSI: float (*funk)(float); */
{
        double x,tnm,sum,del,ddel,b,a;
        static double s;
        static int it;
        int j;

        b=1.0/aa;
        a=1.0/bb;
        if (N == 1) {
                it=1;
                return (s=(b-a)*FUNC2(0.5*(a+b)));
        } else {
                tnm=it;
                del=(b-a)/(3.0*tnm);
                ddel=del+del;
                x=a+0.5*del;
                sum=0.0;
                for (j=1;j<=it;j++) {
                        sum += FUNC2(x);
                        x += ddel;
                        sum += FUNC2(x);
                        x += del;
                }
                it *= 3;
                return (s=(s+(b-a)*sum/tnm)/3.0);
        }
}

#undef FUNC2
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : FUNC2.c                                                 */
/*  Purpose : Computes the value of the function for moment calculation    */
/*-------------------------------------------------------------------------*/

#include <stdio.h>

int n;
double m;

double func2(z)
double z;
{
  double powr,wgt;
  double pow(),W2();

  if (n == 0) {
    powr = 1.0;
  }
  else {
    powr = pow(z,(double) n);
  }

  wgt = W2(z);
  powr *= wgt;

  return powr;
}
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : W2.c                                                    */
/*  Purpose : Compute the Eddington-King Weight function for a particular  */
/*            value of z.                                                  */
/*-------------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>

int n;
double m;

double W2(z)
double z;
{
  double pow();
  double exp();
  double gammln();
  double mm,zm,gamma,gammahold,ezm,mezm,expmz,weight;

  mm = pow(m,m);
  zm = z/m;
  ezm = exp(-zm);
  mezm = -1.0*m*ezm;
  gammahold = gammln(m+1.0);
  gamma = exp(gammahold);
  expmz = exp(mezm-z);
  weight = (mm/gamma)*expmz;

  return weight;
}



double gammln(xx)
double xx;
{
        double x,tmp,ser,gam_return;
        static double cof[6]={76.18009173,-86.50532033,24.01409822,
                -1.231739516,0.120858003e-2,-0.536382e-5};
        int j;

        x=xx-1.0;
        tmp=x+5.5;
        tmp -= (x+0.5)*log(tmp);
        ser=1.0;
        for (j=0;j<=5;j++) {
                x += 1.0;
                ser += cof[j]/x;
        }
        gam_return = -tmp+log(2.50662827465*ser);

        return gam_return;
}
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : GEN_LAMBDAS2.c                                          */
/*-------------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>

#define MAXSIZE 100
#define DEBUG 0

FILE *fp;

void gen_lambdas2(k,A,LAMBDA)
int k;                                  /* maximum size we work with */
double A[MAXSIZE],LAMBDA[MAXSIZE];
{
int i,j,l;
double sum;

LAMBDA[0] = 1.0/A[0];

for (i=1;i<=k;++i)
 {
  sum = 0.0;
  for (j=1;j<=i;++j)
    sum += LAMBDA[i-j]*A[j];
  LAMBDA[i] = -sum/A[0];
  }
printf("\n");
for (l=0;l<k;++l)
  {
    printf("LAMBDA[%d] = %13.10lf\n",l,LAMBDA[l]);
  }

getchar();
}
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : GB_CALC_MOM.c                                           */
/*-------------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>

#define  ZMAX  49
#define  WZMAX  49

int n;
double z[ZMAX],wz[WZMAX];

FILE *ifp,*ofp;


void gb_calc_mom(num_channels,zbar,scale_hgt,moments)
int num_channels;
double zbar[100],moments[100][100],scale_hgt;
{
  int i,k,nfact,cnt;
  int factorial();
  double a,b,zb_sc,nterm,mom,moment;
  double func();
  double trapzd();
  double qromb();
  char infile[15],outfile[15];
  void polint(),nrerror();
  double A[3];

  A[1] = 0.404133;
  A[2] = 0.828590;

ofp = fopen("moments.dat","w");

for(k=1;k<num_channels;++k)
 {
  strcpy(infile,"chan");
  infile[4] = k+48;
  infile[5] = '\0';
  strcat(infile,".dat");

  ifp = fopen(infile,"r");

  for (i=0;i<ZMAX;++i) {
    z[i] = 0.0;
    wz[i] = 0.0;
  }

  for (i=ZMAX-1;i>=0;--i)
    cnt = fscanf(ifp,"%lf %lf",&wz[i],&z[i]);

  fclose(ifp);

  for (i=0;i<ZMAX;++i) {
    z[i] = z[i]/scale_hgt;
    wz[i] = (wz[i]/1000.0)*scale_hgt;
  }

  a = z[0];
```

```
    b = z[ZMAX-1];
    /*zb_sc = zbar[k]/scale_hgt;  The Z_Bar values given by di.c */
    zb_sc = zbar[k];                          /* are already scaled */

    fprintf(ofp,"Channel no. %d\n\n",k+1);
    for (n=0;n<10;++n) {
      nfact = factorial(n);
      nterm = 1.0/((double) nfact);
      mom = qromb(func,a,b,zb_sc);
      moment = nterm * mom;

      /*** Including the Ground - Boundary Conditions for Channels 1 & 2 ***/
      if (k <= 2)
          moment += ((1.0 - A[k]) * pow((-1.0*(zb_sc)),(double)n));

      moments[k][n]=moment;
      fprintf(ofp,"mom[%d] = %lf\n",n,moment);
    }
   fprintf(ofp,"\n\n\n");
   }
  fclose(ofp);
  printf("\n");
  }
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : QROMB.c                                                 */
/*-------------------------------------------------------------------------*/

#include <math.h>

#define EPS 1.0e-6
#define JMAX 20
#define JMAXP JMAX+1
#define K 5

int n;
double m;

double qromb(func,a,b,zb_sc)
double a,b,zb_sc;
double (*func)();
{
        double ss,dss,trapzd();
        double s[JMAXP+1],h[JMAXP+1];
        int j;
        void polint(),nrerror();

        h[1]=1.0;
        for (j=1;j<=JMAX;j++) {
                s[j]=trapzd(func,a,b,j,zb_sc);
                if (j >= K) {
                        polint(&h[j-K],&s[j-K],K,0.0,&ss,&dss);
                        if (fabs(dss) < EPS*fabs(ss)) return ss;
                }
                s[j+1]=s[j];
                h[j+1]=0.25*h[j];
        }
        nrerror("Too many steps in routine QROMB");
}

#undef EPS
#undef JMAX
#undef JMAXP
#undef K
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : TRAPZD.c                                                */
/*-------------------------------------------------------------------------*/

#define FUNC(x,zb_sc) ((*func)(x,zb_sc))

double trapzd(func,a,b,N,zb_sc)
int N;
double a,b,zb_sc;
double (*func)();       /* ANSI: float (*func)(float); */
{
        double fnx;
        double x,tnm,sum,del;
        static double s;
        static int it;
        int j;

        if (N == 1) {
                it=1;
                return (s=0.5*(b-a)*(FUNC(a,zb_sc)+FUNC(b,zb_sc)));
        } else {
                tnm=it;
                del=(b-a)/tnm;
                x=a+0.5*del;
                for (sum=0.0,j=1;j<=it;j++,x+=del)
                  {
                    fnx = FUNC(x,zb_sc);
                    sum += fnx;
                  }
                it *= 2;
                s=0.5*(s+(b-a)*sum/tnm);
                return s;
        }
}

#undef FUNC(x,zb_sc) ((*func)(x,zb_sc))
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : FUNC.c                                                  */
/*  Purpose : Computes the value of the function for the moment calculation */
/*            using the DMSP SSM/T-1 weight functions.                     */
/*-------------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>

#define ZMAX 49
#define WZMAX 49

int n;
double z[ZMAX],wz[WZMAX];

double func(zval,zb_sc)
double zval,zb_sc;
{
  int i,stop;
  double zn,zn_wz,wzval,num,den;
  double fabs(),pow();

/*
zn = pow(zval,(double) n);
return zn;
*/

  stop = 0;
  for (i=0;stop!=1;++i)
    {
      if (zval==z[i])
        {
          stop = 1;
          if (n==0 && zval-zb_sc == 0.0) zn=0.0;
            else zn = pow((zval-zb_sc) (double) n);
          zn_wz = zn * wz[i];
          return zn_wz;
        }
      else if ((zval!=z[i]) && (zval<z[i]))
        {
          stop = 1;
          num = (wz[i]-wz[i-1]) * (zval-z[i-1]);
          den = z[i] - z[i-1];
          wzval = (num/den) + wz[i-1];
          if (n==0 && zval-zb_sc == 0.0) zn=0.0;
             else zn = pow((zval-zb_sc),(double) n);
          zn_wz = zn * wzval;
          return zn_wz;
        }
    }
}
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : GEN_LAMBDAS.c                                           */
/*-------------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>

#define MAXSIZE 100

FILE *ofp;


void gen_lambdas(num_channels,num_moments,A,LAMBDA)
int num_moments,num_channels;
double A[MAXSIZE][MAXSIZE],LAMBDA[MAXSIZE][MAXSIZE];
{
  int i,j,k;
  double sum;
  char st[8];
  FILE *fp;

  LAMBDA[0][0] = 1.0;
  for (i=1;i<=num_moments;++i) LAMBDA[0][i] = 0.0;

  for(k=1;k<num_channels;++k)
    {
      LAMBDA[k][0] = 1.0/A[k][0];

      for (i=1;i<=num_moments;++i)
        {
          sum = 0.0;
          for (j=1;j<=i;++j)
            sum += LAMBDA[k][i-j]*A[k][j];
          LAMBDA[k][i] = -sum/A[k][0];
        }
    }

  for (i=0;i<num_moments;++i)
   {
     printf("\n\n\nLambda No. %d\n",i);
     for (k=0;k<num_channels;++k)
       printf("Channel[%d] = %11.9lf\n",k,LAMBDA[k][i]);
   }
}
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : B2.c                                                    */
/*-------------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>

#define MAX_ZETA 200
#define SCLHT 8.4345

FILE *ifp1, *ifp2, *ofp;

calc_B2(max_der,lambda,R,B,zeta)
double lambda[100],R[MAX_ZETA][100],B[MAX_ZETA],zeta[MAX_ZETA];
int max_der;
{
int z,i;

/* calculate B terms */

 for (z=0;z<MAX_ZETA;++z) {
    B[z] = 0.0;
    for (i=0;i<max_der;++i)
       B[z] += lambda[i] * R[z][i];
  }
}
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : B.c                                                     */
/*-------------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>

#define MAX_ZETA 200

FILE *ifp1, *ifp2, *ofp;

calc_lamb(ch,z,zbs,emlm,lb)
int ch;
double z,zbs[100],lb[10],emlm[100][100];
{
  int i,j;

  for(i=0;zbs[i] < z && i < 7;i++);

  for(j=0;j<9;j++)
   {
    lb[j] = ((z - zbs[i-1]) * (emlm[i][j] - emlm[i-1][j]))/(zbs[i] - zbs[i-1]);
    lb[j] += emlm[i-1][j];
   }
}

calc_B(chan,max_der,lambda,R,B,zeta,zbars)
double lambda[100][100],R[MAX_ZETA][100],B[MAX_ZETA],zeta[MAX_ZETA],zbars[100];
int chan,max_der;
{
int z,i;
int clos_chan;
double lamb[10];

/* calculate B terms */

 for (z=0;z<MAX_ZETA;++z) {
    B[z] = 0.0;
    calc_lamb(chan,zeta[z],zbars,lambda,lamb);
    for (i=0;i<max_der;++i)
       B[z] += lamb[i] * R[z][i];
  }

}
```

```
/*-------------------------------------------------------------------------*/
/*  Program Name : B_T.c                                                   */
/*-------------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>

#define  C1    1.19107e-05  /*  2hc**2 in erg cm**2 sec**-1 ster**-1  */
#define  C2    1.438858     /*  hc/k in cm degK  */
#define  C     3.0e10
#define  MAX_ZETA  200
#define  SCLHT 8.4345

FILE *ifp1, *ifp2, *ofp;

/*-------------------------------------------------------------------*/
/* Temperature calculation using frequencies from the fitted curve */
/*-------------------------------------------------------------------*/

B_T(CHAN,NU,ZB,ZETA,B,T)
double NU[MAX_ZETA],ZB[100];
double ZETA[MAX_ZETA],B[MAX_ZETA],T[MAX_ZETA];
int CHAN;
{
  int i,j;
  double log(),pow(),exp();
  double cv,v3,cv3_B,ln_cv3_B;

/* calculate temperatures */

  for (i=0;i<MAX_ZETA;++i)
   {
    v3 = pow(NU[i],3.0);
    cv = C2 * NU[i];
    cv3_B = ((C1*v3)/B[i]) + 1.0;
    ln_cv3_B = log(cv3_B);
    T[i] = cv/ln_cv3_B;
  }

}
```

```
/*-------------------------------------------------------------------------*/
/* Program Name : NRUTIL.c                                                 */
/* Purpose : Numerical Recipes Utilities file                              */
/*-------------------------------------------------------------------------*/

#include <malloc.h>
#include <stdio.h>

void nrerror(error_text)
char error_text[];
{
        void exit();

        fprintf(stderr,"Numerical Recipes run-time error...\n");
        fprintf(stderr,"%s\n",error_text);
        fprintf(stderr,"...now exiting to system...\n");
        exit(1);
}


   double *vector(nl,nh)
   int nl,nh;
   {
           double *v;

           v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
           if (!v) nrerror("allocation failure in vector()");
           return v-nl;
   }

   int *ivector(nl,nh)
   int nl,nh;
   {
           int *v;

           v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
           if (!v) nrerror("allocation failure in ivector()");
           return v-nl;
   }

   double **matrix(nrl,nrh,ncl,nch)
   int nrl,nrh,ncl,nch;
   {
           int i;
           double **m;

           m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
           if (!m) nrerror("allocation failure 1 in matrix()");
           m -= nrl;

           for(i=nrl;i<=nrh;i++) {
                   m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
                   if (!m[i]) nrerror("allocation failure 2 in matrix()");
                   m[i] -= ncl;
           }
           return m;
   }

   void free_vector(v,nl,nh)
   double *v;
```

```
int nl,nh;
{
        free((char*) (v+nl));
}

void free_ivector(v,nl,nh)
int *v,nl,nh;
{
        free((char*) (v+nl));
}


void free_matrix(m,nrl,nrh,ncl,nch)
double **m;
int nrl,nrh,ncl,nch;
{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}


double *dvector(nl,nh)
int nl,nh;
{
        double *v;

        v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
        if (!v) nrerror("allocation failure in dvector()");
        return v-nl;
}


void free_dvector(v,nl,nh)
double *v;
int nl,nh;
{
        free((char*) (v+nl));
}
```

```
#include <malloc.h>
#include <stdio.h>

void nrerror(error_text)
char error_text[];
{
        void exit();

        fprintf(stderr,"Numerical Recipes run-time error...\n");
        fprintf(stderr,"%s\n",error_text);
        fprintf(stderr,"...now exiting to system...\n");
        exit(1);
}
```

References

[1] S. Bochner and W. Martin, **Several Complex Variables**, (Princeton University Press: Princeton NJ), (1948).

[2] M. Born and E. Wolf, **Principles of Optics**, 6th edition, (Pergamon Press: Oxford), (1959).

[3] P. A. M. Dirac, **The Principles of Quantum Mechanics**, (Clarendon Press: Oxford, U. K.), (1947).

[4] A. S. Eddington, "On a Formula for Correcting Statistics for the Effects of a Known Probable Error of Observation", *Monthly Notices Roy. Astron. Soc.*, **73**, 359-360, (1913).

[5] A. Erdelyi, **Operational Calculus and Generalized Functions**, (Holt, Rinehart, and Winston: New York), (1962).

[6] R. G. Hohlfeld, J. I. F. King, T. W. Drueding, and G. v. H. Sandri, "Solution of Convolution Integral Equations by the Method of Differential Inversion", *SIAM J. Appl. Math.*, **53**, pp. 154-167, (1993).

[7] E. Isaacson, and H. B. Keller, **Analysis of Numerical Methods**, (Wiley: New York), (1966).

[8] J. I. F. King, "Theory and Application of Differential Inversion to Remote Temperature Sensing", in **Advances in Remote Sensing**, A. Deepak, H. E. Fleming, and M. T. Chahine, eds., (A. Deepak Publ.: Hampton, VA), 1985.

[9] J. I. F. King, R. G. Hohlfeld, and J. C. Kilian, "Application and Evaluation of a Differential Inversion Technique for Remote Temperature Sensing", *Meteorology Atmos. Phys.*, **41**, 115-126, 1989.

[10] M. J. McFarland, R. L. Miller, and C. M. U. Neale, "Land Surface Temperatures Derived from the SSM/I Passive Microwave Brightness Temperatures", *IEEE Transactions on Geoscience and Remote Sensing*, **28**, 839-845, 1990.

[11] J. G. Mikusiński, **Operational Calculaus**, (Pergamon: Elmsford NY), (1959).

[12] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, **Numerical Recipes in C**, (Cambridge University Press: Cambridge, UK), (1988).

[13] B. A. Rhodes, "Integral Equations and Inverse Problems", Ph.D. thesis, Boston University, (1991).